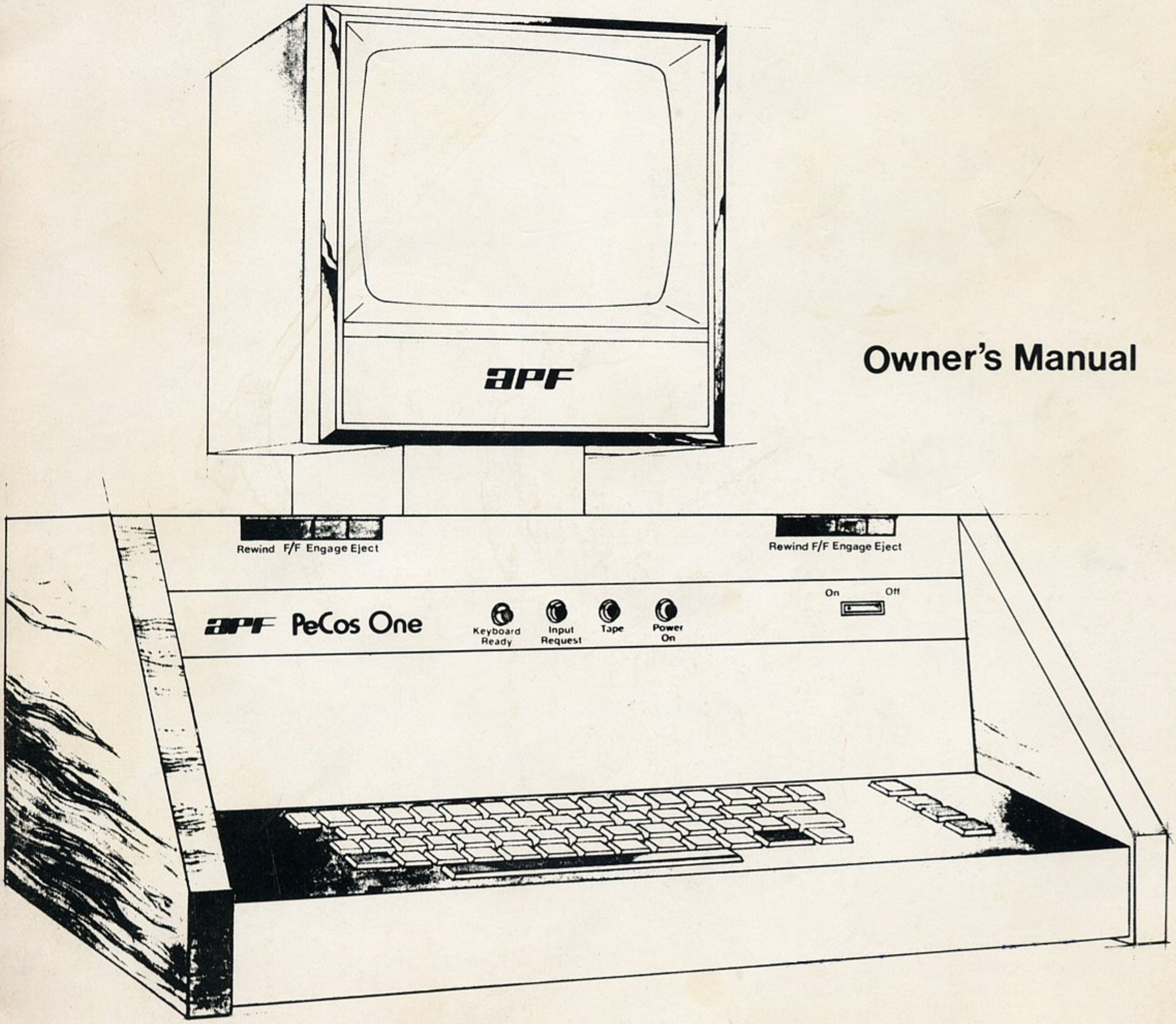
apf

An Introduction to PeCos One

The Personal Computer



Owner's Manual

apf ELECTRONICS INC., 444 MADISON AVENUE, NEW YORK, N.Y. 10022 (212) 758-7550

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

A new era in personal electronics is dawning. It's been dreamed of--and talked about for years. Now it's here. It's the Personal Computer. Your APF, PeCos One, utilizes recent developments in semiconductor integrated circuitry, to provide you with a powerful, computational instrument, capable of solving a wide variety of problems.

Till now, the computer language barrier had clearly been the major problem in translating the myriad capabilities of the computer into a meaningful product for personal use by consumers. The advanced PeCos One system is launching a new breed of computer, which talk to the user in a non-technical, easily-mastered language-English. The computer language used in PeCos (Personal Computer System), is actually a variation of the popular JOSS , language developed by the prestigious Rand Corporation, for those who needed direct access to a computer, but had neither the time nor interest to learn conventional, and highly complex-computer language. APF, has refined its version of JOSS even further.

With this language, you can converse with the system, in simple imperative English sentences, (made up of PeCos vocabulary), always using correct grammar, and punctuation, (PeCos will insist on that). If you make a mistake in entering data, or asking questions, PeCos will politely inform you of the mistake--and what to do about it--in simple English, through a unique, "Error commentary", system.

In the future there will be additional user programming manuals available.

You can write to us now, and we will forward information to you when available.

If you feel that you have been successful in mastering, and programming PeCos, we are interested in hearing about any programs written on PeCos

SEND ALL CORRESPONDENCE TO:

APF ELECTRONICS, INC.
DEPARTMENT P1
444 MADISON AVENUE
NEW YORK, N.Y. 10022

SPECIFICATIONS

MAIN UNIT	CRT (Monitor), Keyboard, MPU, I/O, 2 Cassette Decks, Power Supply, Memory RS232-C, (Output only).
OPTIONS:	2 Additional Cassette Decks, Printer
MICROPROCESSOR:	6502
KEYBOARD:	60 Keys, Durable, typewriter style construction
CRT:	9-inch, Black and White, 40 characters per line, 16 lines total, Upper and lower case.
POWER:	120-Volt, 50/60 Hz
CASSETTE:	Standard Audio Cassette Drives with motor control by computer .Manual rewind and fast forward.
CASSETTE FILES:	Up to 4 tapes addressable. Semi-automatic control makes files on tape addressable by tape number, Search can begin at any position on, tape. Files also accessed by name.
BAUD RATE:	800 (speed-tolerant recording)
ROM MEMORY:	24K, PeCos Interpreter, and operating system.
RAM MEMORY:	16K

Chapter 2

GETTING STARTED

Unpack the system carefully, remove all packing material and make sure you locate all cables, literature, etc. Save the packing material and carton in case you have to transport the system.

To power-up PeCos, follow the steps below:

1. Place the console in some spot in which it will be comfortable to sit and type into the keyboard.

2. Make sure the back of the unit is not blocked, there are vent holes here to allow air into PeCos, and should not be blocked. Similarly, there are vent holes on the underside of the cabinet and these should also not be blocked.

3. Make sure the PeCos power switch is turned off.

4. Connect the main power cord, to the wall socket this is a three prong type plug. If your outlet has only two prongs then use a 3-2 plug adaptor, and make sure to connect the ground wire to ground, (such as the screw that holds the outlet plate on).

5. Connect the other end of the main power cord to the A.C. plug in the rear of PeCos.

6. Connect the power cord of the monitor to the two prong auxillary A.C. socket in the rear of PeCos.

7. Connect the video cable from the rear of PeCos to the rear of the monitor.

8. Turn the power switch of the monitor on and wait about 30 seconds till it warms up.

9. Turn the power switch of PeCos on, in about 2 seconds the monitor should read:
   "PeCos Here".
   Also, the power light and keyboard ready light should be on.

10. If the above does not happen, repeat steps 3-9.

FRONT PANEL

The front panel consists of a power switch and 4 red lights.

POWER SWITCH-The power switch is located on the right front.

LAMPS:

1. Power-The power lamp will be lit whenever the power switch is turned to on.

2. Tape-The tape lamp is on when PeCos is reading or writing to any of the tape decks. Due to the PeCos method of hand-ling the tape system the light will actually "blink" on and off when the tapes are being read from or written to.

3. Input Request-This lamp will light when PeCos is waiting for an input from the keyboard because some part of the program has requested an input.

4. Keyboard Ready-This lamp will light when PeCos is not running a program or executing a command and is ready for you to input a new statement.

CRT MONITOR

This is used by PeCos to display information to you. When you key in sentences or data, they immediately appear on the monitor. The monitor can display any of 93 alphabetical, numerical or-special symbols. Each of these is called a character). It can display both upper and lower case letters. The screen is capable of displaying up to 640 characters at one time, this consists of 16 lines with 40 characters each.

PeCos fills information to the screen from top to bottom. When the bottom line is filled up, PeCos does what is called scrolling-that is it shifts the top line off the screen, the next 15 lines move up and adds a new line to the bottom. The rate at which this scrolling occurs is controllable from the keyboard

section. (Slower, Faster, Hold).

CRT MONITOR-Continued

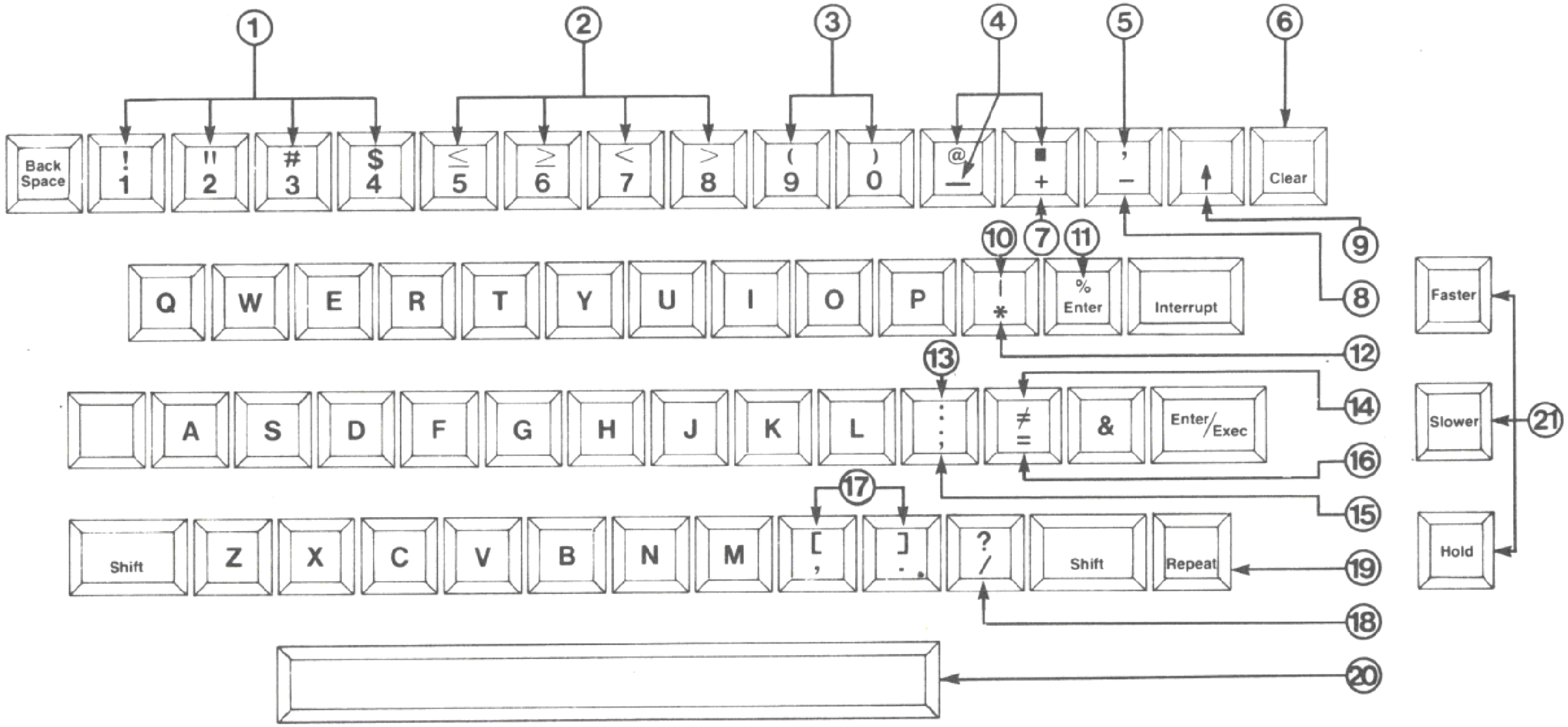
An example of this is if PeCos is displaying a group of names and addresses you can have it scroll fast until it displays the area you are looking for, then slow it down, and finally hold the scrolling.

CRT CONTROLS-The monitor has adjustment controls, Brightness, Contrast, Vertical Hold, and Horizontal Hold. These serve the same functions as on any standard television.

INTERNAL STORAGE-To allow storage of programs or information PeCos has an internal Memory System. The smallest amount of storage space is called a PEC. The basic PeCos system has 1864 PECs of internal storage. Storage to or retrieval from this memory is instantaneous. When power is turned off, any thing stored in this memory is lost.

TAPE DECKS-There are two tape decks included with PeCos. They can record, or read using standard audio type cassette tapes. The tape decks provide a means of saving programs and data. They also, can hold much larger programs or data than is possible in the internal memory.

KEYBOARD



① PUNCTUATION SYMBOLS

② LOGICAL OPERATORS

③ PARENTHESIS

④ SPECIAL SYMBOLS

⑤ APOSTROPHE

⑥ LINE CLEAR

⑦ ADD

⑧ SUBTRACT

⑨ EXPONENT

⑩ VERTICAL BAR

⑪ PERCENT

⑫ MULTIPLY

⑬ COLON

⑭ NOT EQUAL

⑮ SEMICOLON

⑯ EQUAL

⑰ BRACKETS

⑱ DIVIDE

⑲ REPEAT ENTRY

⑳ SPACE

㉑ SCROLLING CONTROLS

KEYBOARD

The keyboard is used to "speak" to PeCos. The keyboard con-sists of 60 keys, and is layed out similar to a typewriter.

SHIFT KEY-By holding the shift key down, and then simultaneously touching any dual function key, the upper function will be entered.

> *NOTE: There is no shift lock, therefore, you must hold the shift key down while you press the desired function.*

ALPHABET-a-z obtain lower case letters by just touching

the key down. To obtain upper case letters, press the shift key and the letter together.

NUMERAL-1-9 and 0 Note how the 1 and 0, are written to distinguish them from "I" and "O".

As an example to see how the keyboard works, type in the following: The sly, quick fox jumped over the lazy brown dog.

0 123456789.

As you press a key you should see that letter or character appear on the monitor.

Also, you will notice a white square on the screen that winks at you. This is called the cursor. As you type in from the keyboard, the cursor moves its position to indicate when the next character will be displayed.

ENTER/EXECUTE: This is a very special key to PeCos. When you press this, it tells PeCos to look at what you have typed in. In the above example, PeCos has not looked at or inter-preted what you typed in. Whenever, you want PeCos to take over control and interpret what you typed in, press Enter/Execute.

If you now press Enter/Execute, PeCos will respond with
Eh?
This is one of PeCos' many comments when you do something it does not understand. You typed in and entered a statement that PeCos can not interpret. Don't worry, you haven't hurt PeCos. If you have some patience till we reach Chapter 5-11, you will then read all about the proper way to speak with PeCos.

PUNCTUATION SYMBOLS

The shifted punctuation symbols are:

! " # $ @ ' ? :

To type these in, just touch the shift key, and the appropriate punctuation symbol key simultaneously.

The unshifted punctuation symbols are:

,.;

To type these in, just. touch the appropriate punctuation symbol.

MATHEMATICAL FUNCTIONS-All are non-shifted functions. + : Used to indicate algebraic sign to a number or to indicate addition.

- Used to indicate algebraic sign to a 'number or to indicate subtraction.

* Used to indicate multiplication-Since we usually use an X for multiplication, but X is also a letter, we use the * symbol to represent multiplication--ex 2 times 4 is typed as. 2*4.

/ Used to indicate division--ex 6 divided by 2 is typed in as: 6/2.

Symbol for exponent ex-10 to the 5th power (10^5) is enteredas:10^5

GROUPERS-There are 4 symbols that can be used for grouping:

(	Left parenthesis
)	Right parenthesis
[	Left bracket
]	Right Bracket

Note: These must always *be used in pairs.*

RELATIONAL OPERATORS

The symbols below are called Relational or Comparison operators. You can do comparison of numerical expressions or textual expressions.

<	Symbol for less than
>	Symbol for greater than
≤	Symbol for less than or equal
≥	Symbol for greater than or equal
≠	Symbol for not equal
=	Symbol for equal

REPEAT KEY-By touching the repeat key, and any key, (except clear, or enter/execute), the entry will keep repeating. That is if you touched repeat and the "a" key, the letter "a" would keep being entered until you released either the repeat key or "a" key. If you want to keep repeating a shifted symbol, touch shift, repeat and the key simultaneously.

INTERRUPT-If a program is in progress, and you want to stop it, you can do so by pressing the Interrupt key. PeCos will halt, display where it stopped, and give a message that it was stopped by an interrupt. There is an interrupt disable switch on the rear panel. If you wish to disallow interrupts this switch can be set to do so.

■ TOUCH SHIFT KEY AND + KEY

The solid square has two purposes:

1. Can be used as textual symbol with no special meaning to PeCos.
2. Can be used in a display statement. In a display statement it means erase the screen.

% KEY- This key is used to enter the percent symbol. To PeCos It does not cause a calculation, but just enters a % symbol.

| VERTICAL BAR-TOUCH SHIFT'KEY AND * KEY

1. Used to indicate function for absolute value of a numerical expression. You must use the vertical bar in pairs. (See Functions).
2. Can be used as a textual symbol with no special meaning to PeCos.

SCROLLING CONTROLS

To control the rate of scrolling on the monitor there are 3 controls.

Faster-Increases the rate of scrolling. Each touch of the faster key decreases the time between line scrolls by 1 second. Note: The scrolling speed is also internally fixed by the time it takes to do a calculation. The maximum scroll rate is 2 lines/second.

Slower-Decreases the rate of scrolling. Each touch of the slower key increases the time between line scrolling by 1 second.

Hold-By touching the hold key, scrolling stops completely. If you release the hold key, scrolling returns to the previous set rate.

MISCELLANEOUS SYMBOLS-

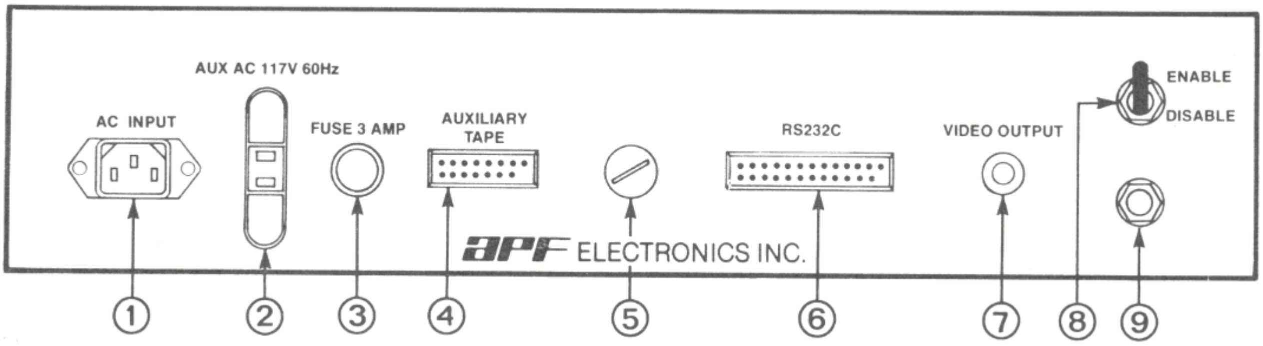
_:Underscore. An Underscore serves the following purposes:

1. Can be used as a textual symbol with no special meaning to PeCos.
2. Can be used in a form statement to define fields.
3. Can be used in a display statement. In this case it causes PeCos to display a blank line.

REAR PANEL

Figure 2 shows a diagram of the rear panel of PeCos. A description of each plug, socket, and switch is given below.

REAR PANEL



① MAIN AC RECEPTACLE

② AC OUTLET FOR CRT MONITOR

③ 3 AMP FUSE

④ 15 PIN SOCKET FOR OPTIONAL TAPE DECKS

⑤ BAUD RATE

⑥ RS232 TRANSMIT SOCKET

⑦ OUTPUT PLUG FOR CRT MONITOR

⑧ INTERRUPT LOCKSWITCH

⑨ SYSTEM RESET BUTTON

1. MAIN A.C. RECEPTACLE-This is a 3 prong A.C. Receptacle The main A.C. for PeCos comes into here, make sure the main A.C. cord is pushed in all the way. It is designed for a tight fit to avoid the possibility of the plug slipping out while you are using PeCos.

2. AUXILLARY A.C. OUTLET-This is a 2 prong A.C. output connector. As long as the main A.C. cord is plugged in, there will be power at this point, (whether the power switch is on or off). This outlet is to be used power for the CRT Monitor.

3. FUSE-There is a Fuse used to prevent overloads. If you have to replace the fuse,
Use only 3 AMP,Slow Blow fuse.

4. OPTIONAL TAPE DECK SOCKET-You can obtain an additional 2 tape decks (Model MPT-10). This socket is used to connect them to the main console.

5. BAUD RATE SWITCH-This sets the rate of transmission from the RS232C plug. There are 3 rates (110, 300, 1200) and you must set the proper rate for a printer that is connected.

6. RS232 C TRANSMIT PLUG-This plug is used to connect up an optional printer. The pins are wired for standard RS232C signals.

7. OUTPUT PLUG FOR MONITOR-This plug (called a phono type plug), connects to the input of the CRT Monitor. There is a cable provided for this purpose. This must be connected for the CRT to display PeCos messages.

8. INTERRUPT LOCKSWITCH-In the enable mode, this switch will enable the touching of the interrupt key, to interrupt PeCos. If this switch is in the disable mode PeCos will not respond to your touching the interrupt key.

9. SYSTEM RESET-Pressing this button, has the same effect as turning power off, and on. System initialization occurs all memory is cleared, and the CRT screen will say "PeCos Here".

Chapter 3

COMMUNICATION WITH PECOS

Interaction between man, and machine has always been a problem. All machines always require that we follow a set procedure of rules to make them work. As an example, to use the telephone to call Mr. Jones-1) Pick up the receiver-2) Wait for dial tone 3) Dial area code 4)Dial the number-disobey any rule and you won't speak to Mr. Jones.

This procedure can be called a program. To program PeCos there is a specially designed language--called PeCos. PeCos, is derived from a frequently used language, JOSS. JOSS is a trademark of the RAND Corporation. It has been implemented many times and is well known although under numerous names.

The basic of the PeCos language is to allow the user to give direction to PeCos by keying-in imperative English type sentences. The structure of these sentences is the same as English sentences. They contain verbs, nouns, clauses, etc, and there is a simple set of rules for forming these sentences.

PeCos' vocabulary consists of about 40 words, (called key words). Of course, 40 words would be insufficient for good *communication* so you can put in other words, and terms by defining expressions and formulas to PeCos.

What happens when you violate one of the rules of use of PeCos. Many computers get very touchy about this, and refuse to continue or even tell you simply what you did wrong. Not so with PeCos; when you violate one of PeCos's rules, it will try to tell you in plain English what you did wrong.

An example is, if you key in:

Do part 37. *(and there is no part 37 stored in PeCos)*

I can't find part 37. *(PeCos will respond with)* Further more, in most instances, PeCos will allow you to correct you mistake, and then it continues on with its task.

Chapters 5-10, of this manual explains the details of the PeCos intelligence.

It is suggested, you read this section carefully through.

It is your key to getting the most out of PeCos.

Chapter 4

A WORD ON PROGRAMMING

PeCos, is designed for people who have no experience in programming, or working with computers; as well as experienced computer users. This section is mostly for the non-experienced user.

To get some usefullness out of PeCos, you must give it a "program", to do. A program, is an orderly group of steps that PeCos is to perform, and when it has finished it will give you the result you wanted.

PeCos is very good at three things:

1. Following the instructions you give it.

2. Doing calculations far faster than is humanly possible.

3. Filing items and being able to find them. How

do you write this program?

1. First you must clearly define to youself what the problem is, what information do you have available and what is the procedure to go about solving the problem. Sometimes it is best to write this down on paper.

2. Define what the main task is, and what the subtasks are. From this you can generate a "flow chart", or problem "tree". Indicate which subtasks must precede which others, list formulas, and abbreviations.

3. When you have completed this flow chart, you are ready to start entering the program into PeCos by keying-in PeCos sentences.

PeCos will assist you in getting statements in so it can understand, and perform them. When you enter something PeCos doesn't understand, PeCos will try to tell you in plain English what you did wrong. You can usually make the necessary correction and then have PeCos continue.

BUGS:

You've done all of the above, run the program and you just know the answer PeCos is giving you has to be wrong. You've got what is commonly known to programmers as a "bug". Somewhere you've told PeCos to do a step or part that will cause the answer to be wrong, or you've not told PeCos to do a step or part that is necessary to get the right answer.

Below is a procedure to try to find these bugs:

1. Review your flow chart. Did you specify the procedure correctly?
2. Go through the program manually using a few initial values.
3. PeCos could be performing a command in a different way than you expect. It is important that you understand exactly how all the commands work. Try some practice examples to see if PeCos does what you expect it to do.

4. Since PeCos programs are broken into parts it is very easy to have it do just certain parts, or steps of the program to see if it is doing it correctly.

5. You can add a step anywhere Very easily, (and later delete it just as easy). May-be you want to add a stop statement in the middle of a part. Then let the pro-gram run, and it will stop at a certain step at which time you can see what PeCos has done.

Sometimes, it can take a lot of time to get a program written and working properly, but once you have finished,you can then have the benefits of that program.

In writing programs, go slow, be careful, and review carefully what you are doing. Most important read this manual thoroughly.

Chapter 5

DEFINITIONS AND RULES

Before proceeding with the details of the PeCos language, it is necessary to review, and establish certain definitions.

EQUATIONS, VARIABLES, AND EXPRESSIONS

To do problem solving, we must give PeCos an equation to follow. An equation, simply states the procedure to be followed to find a result. As an example, the equation to de-termine the amount of simple interest collected on money placed

in a bank is: Interest is equal to amount of deposit,
multiplied by the interest rate.

In dealing with a computer, it is more convenient to use symbols, and operators to state an equation. Let's first get two more definitions, and then rewrite the equation for interest.

A variable,is something whose value can change, (or vary); such as the interest, interest rate, or amount of deposit can vary, and are thus called variables.

A symbol, is a short cut way of designating a variable or operation. *(Example: the symbol for addition is +).*

With PeCos, you can use any of the 52 upper or lower case letters as a symbol to designate the name of a variable. It's usually a good idea to choose letters that remind you of the full name of the variable . Now let's rewrite the equation for interest using symbols.

_i=r*d._

i-is the symbol for interest

r-is the symbol for rate

d-is the symbol for deposit

*-is the symbol for multiplication

=-is the symbol for stating that what is on the left side is equal to what is on the right side.

Expressions-An expression is a representation using symbols, such as r*d. Expressions are evaluated to result in some value. An equation is one type of expression, but in PeCos there are three types of expressions allowed: _Numerical, Logical, or Textual._

Numerical-consists of only numbers. The evaluation of a numerical expression is always a number. An equation is a numerical expression.

Textual-A textual expression is anything enclosed in a pair of quotes, (except a quote). The result of the evaluation of a textual expression is simply the text itself. Logical-consists of logical (or relational) operators. (< , > , ≤ , ≥ , =, ≠, true, false, or, not, and). The evaluation of a logical expression is either true, or false.

1. Numerical expressions

3*5+2 2*i+sqrt(a) r*d	*Numerical expressions must only contain numbers or variables that are numerical*
2*"cat" (a<b)+6	*Do not mix a textual and numerical value, or logical and numeric values*

2. Textual expressions

"textual statement" "2+6-3" "text is any characters in quotes-2+6-3↑4"	*anything in side a pair of quotes, (except a quote) is treated as a text string.*
"Do not leave out an end or beginning quote"	*but don't forget both quotes*

3. Logical expressions

1=2 1<2>0 true false 6≠3+1	*logical expressions use relational operators* *also the words, true or false*
"aa">"ab" "cat"=1+2	*they show relations between any types of expressions and the evaluation is always either true or false.*

Expressions, play a very important role in PeCos. When we enter statements, we will give PeCos expressions which it will evaluate. It will use the values of these expressions and maybe display, print, or use them in other expressions, or will do anything we tell it to do with them.

RULES OF PRECEDENCE

What are the rules PeCos will follow in evaluating expressions. For simple expressions such as 1+6, or 3<4. It is obvious how PeCos evaluates these and what there results are. But what about complicated expressions such as:

```
3*4+6-9/8↑12*sqrt(13-a*(b+2)).
```

<u>Numerical Expressions:</u>

PeCos follows conventional rules in determining the order of evaluation of numerical expressions. Expressions are evaluated from left to right as follows:

1. Exponentiation done first.
2. Then uniary plus/minus (algebraic sign)
3. Then multiplication/division
4. Then addition/subtraction
5. Groupers,(parenthesis/brackets) *of course, can alter precedence rules.*

The number of groupers is practically unlimited, and so in complicated cases the best advice is to be liberal with parenthesis/brackets, and to do a sample side calculation with numbers, to see if you are getting what you want. Many a good program has foundered for lack of attention to precedence.

Groupers-continued

The only two things to remember with groupers is that:

1. The number of left parentheses, must be equal to the number of right parentheses, (and the same is true for brackets).

2. A left parenthesis can't be matched with a right bracket, (and vice versa).

Examples of how PeCos would evaluate numerical expressions are listed below:

Expression	Evaluation	Explanation
-2↑2	-4	*2↑2 first, then -.*
1+1/2	1.5	*1/2 first, then +.*
(1+1)/2	1.	*expression in parenthesis first, then division.*
1/3*3	.999999999	*division and multiplication, have same precedence so evaluate from left to right.*
2↑3↑2	64	*evaluate left to right 2↑3 first, then 8↑2.*
2↑(3↑2)		*do expression in parenthesis first.*
2↑(3*(4-2))+3↑2	73	*do inner most grouped operation, (4-2), then next grouped operation, (3*2). When all grouped operations are finished, evaluate left to right, and do exponenation first (2↑6), then 3↑2. Do addition last.*

Textual Expressions

1. The evaluation of a text expression is simply text itself.

2. The only character not permitted in a text string is a quote.

3. A null string, consists of no characters.

Examples of textual expressions,and their evaluations are listed below:

Expression	Evaluation	Explanation
"cat"	cat	*anything in quotes*
"123+456"	123+456	is *text, and is pre-*
"any group of letters		*served as it itself.*
numbers of symbols		
up to 80 characters"		*but don't place a*
"123 "+456"	illegal	*quote in the middle.*
		a null string con-
“”	null	*sists of no characters.*

Logical Expresions:

1. Relational operators, (<, >, ≤, ≥, =, ≠) are used to compare numerical expressions, or textual expressions to each other.

2. Boolean operators, (or, and, not) are used to compare logical expressions, to each other, and also produce a logical expressions.

3. The evaluation of a logical expression is always either true, or false.

4. In evaluating logical expressions, PeCos:

   a. Evaluates from left to right.
   b. All arithmetic operators are evaluated first.
   c. Then relational operators.
   d. Then "not".
   e. Finally "or", "and".

5. Groupers, (parenthesis/brackets) can of course alter precedence rules.

Some examples of how logical expressions would be evaluated are below:

Expression	Evaluation	Explanation
1<-2	false	*Do* numeric *operators* first, then *logical*
1+2>6-5	true	
3-2=6	false	*Equals sign is a relational operator*
"cat"="dog"	false	*Text expressions can be compared*
A>B	false	
1<-2 or 3-2=6	false	*The words or,* and not *are known as Boolean operators, and com-pare logical expres-sions.*
"cat"="cat" and A>B	true	

ARRAYS-Although many programs only require several variables others require hundreds or thousands. Since PeCos only allows the use of the 52 upper or lower case letters as symbols of the names of variables; what do we do when we need hundreds or thousands of names?

We use what is called *Arrays, or subscripted variables.* An array is simply an arrangement of 1, or more values, grouped together using a common symbol as their name.

As an example, suppose we have 10 employees in a company. We could use different letters to represent the salary of each employee, (ie: a=employee 1's salary, b= employee 2's salary, etc.). Another way is to use an array, in this case we would use only one common letter, and associate different numbers with this let-ter, we then have 10 different letter-number variable names.

Arrays-Continued

Each will have as it's value, an individuals salary. Let's use the letter E, as the name of this array, and then the 10 letter-number names are:

E(1) , E(2) , E(3) , E(4) , E(5) , E(6) , E(7) , E(8) , E(9) , E(10). E is called the array name, and the numbers are called the subscripts. They are read as E sub 1, E sub 2, etc. The subscripts are placed in parenthesis, following the name of the array. PeCos is very powerful when it comes to dealing with arrays.

The rules PeCos follows with arrays are:

1. The name of an array, is a single upper or lower case letter, (just like any other variable name).
2. An array, can have from 1 to 10 subscripts (Not just 1 as in the example).
3. A subscript must be a whole number (an integer), between -999 and +999.

Example:

a(1,3)-Array named a with two subscripts. First subscript is 1, second is 3.

b(1,3,6,126)-Array named b, with 4 subscripts.

c(1000,-22)-Illegal element of array since first subscript is greater than +999.

GENERAL RULES OF FORM OF PECOS STATEMENTS

PeCos statements are written as normal English type imperative sentences. The rules of sentence structure are as follows:

1. The first word of any sentence must be a *Command Verb.* It must start with a capital letter.

2. There can be only one command verb per sentence.

3. All sentences must end in a period.

4. A sentence can have up to 80 characters, including the period at the end .

5. Use proper punctuation, spacing, and spelling in all sentences.

6. Spaces (or blanks) may be used freely except:

   a. within numbers
   b. between the name of a function, or array, and the left parenthesis.
   c. within key words

7. Some sentences have only a command verb.

8. Some sentences have a command verb, and a noun (or more than one noun).

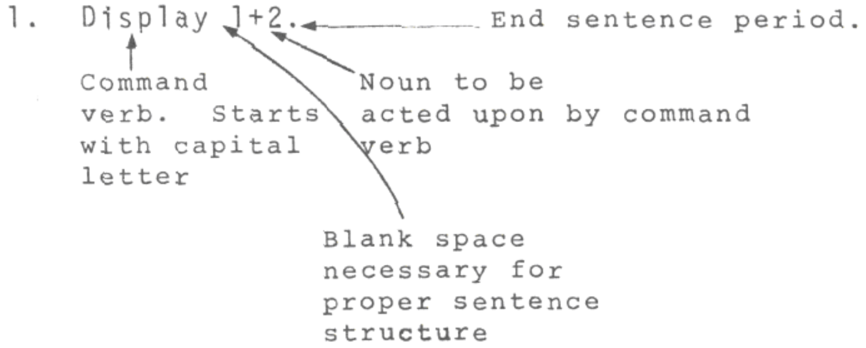
9. All sentences may have an optional if clause.

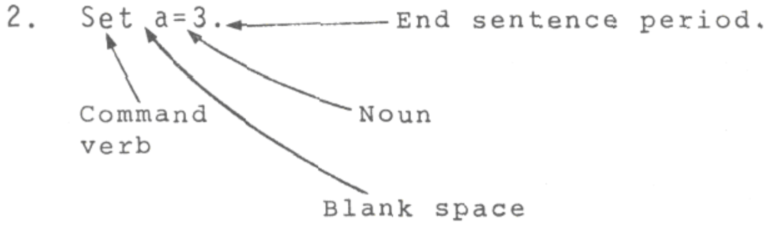
The general form of a PeCos statement is:

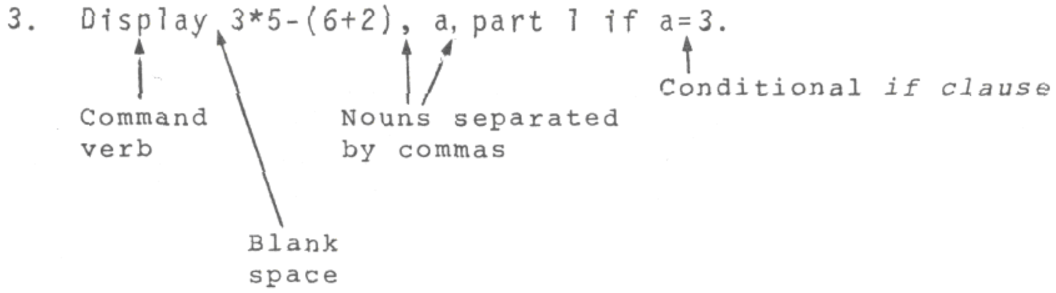
Verb______	Noun 1, Noun 2, Noun 3	_________	.
must start	Nouns: certain sentences	optional	end of
with a command	have none, other can have	modifiers	
	sentence		
verb, with a	many, Nouns are separated	or conditions	
period			
capital first	by a comma		
letter			

BELOW ARE SOME SAMPLES OF PROPER PECOS SENTENCES

Note: *After keying in a sentence always touch enter/execute so PeCos will look at the sentence, interprete and perform.*


1. Display 1+2.
End sentence period.
Command verb. Starts with capital letter
Noun to be acted upon by command verb
Blank space necessary for proper sentence structure




3. Display 3*5-(6+2), a, part 1 if a=3.
Command verb
Nouns separated by commas
Conditional *if clause*
Blank space

4. Erase. End of sentence period.

Command
verb

In the above there is no noun, the noun is the CRT screen and is assumed.

5. Do part 1 for i=1, 3, 5.

Command noun condition End sentence period
verb

PECOS LANGUAGE KEY WORDS

Listed below, are the key words of the PeCos language. These are English words that PeCos recognizes. Details of what these words do follow:

VERBS	MODIFIERS	NOUNS	FUNCTIONS CONDITION
Display	in form	steps	ip-integer part
Set	for	parts	fp-fractional part
Do	if	all	xp-exponent part
To	on	timer	dp-decimal part
Demand	from	size	sgn-algebraic sign
Done		true	sqrt-square root
Cancel		false	log-natural log
Delete		record	exp-exponentiation
File		form	sin-trigometric sine
Recall		null	cos-trigometric cosine
Format		variables	arg-argument
Read		expressions	-absolute value
Write			or-logical or
Erase			and-logical and
Label			not-logical
not Find			
Go			
Print			

EDITING

Line editing must be done before a sentence is entered into PeCos, via the Enter/Execute key'

Backspace (BS)- You can use the backspace key which will move the cursor backwards, (you can move back as far as the first character of the line).

If you see an error before entering the sentence, use the backspace key to move the cursor to the error. Then retype the rest of the sentence correctly.

Clear-The clear key, will move the cursor from its present position, to the first character position of the sentence, It will delete anything entered on that line.

Example

Displat 1+1. *Keyword display is misspelled. Use the backspace key to move the cursor to the t, then retype the rest of the sentence correctly.*

DIRECT AND INDIRECT STATEMENTS

A PeCos statement, (or sentence) can be given in either a direct or or indirect mode.

A Direct Statement, is acted upon by PeCos immediately after it is entered. PeCos does not keep a record of the statement in it's memory. It simply interprets the statement and performs it.

If you want a direct statement performed again, you must re-enter it. A direct statement always starts with a verb.

EXAMPLES

Set a=3+4.
Display a, 4+9,
a=7.
4+9=13.

Direct statement is immediately interpreted and acted upon.

Display 6/(a-a).

Errors are immediately found.

I have a zero divisor

Set a=123456789234.

Please limit numbers to 9 significant digits.

INDIRECT COMMANDS-Stored Statements

PeCos allows you to store statements. These statements are not executed by touching Enter/Execute but will be interpreted and executed only when a direct command to do them is given.

Note:*Since they are not interpreted when entered, errors will be found only when they are executed.*

To store commands, start the sentence with a number-the step number. The number contains an integer, (to the left of the decimal point), and a decimal fraction to the right of the decimal point).

The integer is called the part number, and the fraction is called the step number.

Example

2.01 Display 1+2.

The above statement is stored as step 01 of part 2.
The restrictions on part/step numbers is that the total number of digits must be 9 or less and it must be positive.

Examples of legitimate step/part numbers are

111.229998 Set a=1+2.   *part 111, step 229998*

13457.001 Set a=1+2.   *part 13457, step 001*

1.1 Set a=1.2   *part 1, step 1*

Illegal examples are

-1.1 negative numbers are not allowed

123.5566778 more than 9 digits

19999999.41

A sequence of steps with the same number to the left of the decimal point is called a part, identified by the integral number. Steps are stored in the sequence of increasing step numbers. PeCos reorders your input according to these step numbers, regardless of the order of inputting. If a step number is used that was previously used, then the latest entry will replace the old one.

Display of Programs

In order to see what your program looks like it is necessary to command PeCos to do just that. This can be accomplished by displaying steps, several steps, parts, several parts or all parts. This is useful when editing or just reviewing your programs.

Examples

```
A). 1.1 Display 2+2.            Stored (indirect) command
                                Part 1, step 1.
B). 1.2 Display 3*4.
                                Order of entry of step
    1.05 Display 5/6.           does not have to be sequential
                                step 1.05 will be placed before 1.1

    Display part 1.             PeCos reorder steps into ascending
                                order
    1.05 Display 5/6.

    1.1 Display 2+2.

    1.2 Display 3*4.


    2.1 Display 4<5.            Part 2, step 1


    1.2 Display 3*4*5*6         Step 1.2 replaces previous step 1.2


    Display part1, part 2.

    1.05 Display 5/6.

    1.1 Display 2+2.

    1.2 Display 3*4*5*6.

    2.1 Display 4<5.
```

COMMENT STATEMENTS

A step in a program can be used as a comment line by placing the symbol * as the first, (after the step number), or last symbol in a statement. It is useful to put comment statements into a program to describe what the program is doing. PeCos, stores a comment statement exactly as you put it in.

Example:

```
1.1* This program sets x and y equal to 2 and displays
1.2* them
1.3 Set x=2.
1.4 Set y=2.
1.5 Display x, y.
```

ORDER OF PROGRAM EXECUTION

The part, is the major unit of program execution. When commanded to *do* a *part,* PeCos follows the step sequence within that part, unless directed by a *step* to go to some place, *(a step, or a part),* other than the next step.

For the verbs that change the normal sequence, see *Do, To, Done,*

To get a stored program to be interpreted, and executed, you must always give a direct command. (Such as a Do statement) If you typed in:

Do part 1. *Part 1 would be executed.*

Chapter 6

Display

The verb display, is used to have something to be displayed on the CRT screen. The general form of a display statement is:

```
Display       Noun 1, Noun 2, Noun 3...
```

Nouns can be displayed
each noun is separated by
a comma from other nouns
There can be 1 or more
nouns, in a display statement

Optional modifier or
conditional field.
Maybe an *if clause*
or *in form clause.*

The nouns in the display statement can be any of the allowable nouns of the PeCos language. Let's try some examples of a display statement. Reminder: *After typing in a statement, touch Enter/Execute, so PeCos will interpret what you typed in. Through out the rest of this manual, Enter/Execute will be abbreviated by E/E.*

A. Math Calculations

If we type in

```
Display 1+1. E/E
```

The screen will show

```
1+1=2.
```

In this example, we have used an expression as the noun in the display statement. What PeCos has placed on the screen, is the expression on the left side, one equal sign, evaluation of the expression of the right side.

Now let's try more than 1 noun, in a display statement.

```
Display 1+1, 6-2, 5*3, 6/8. E/E
            1+1=          2
            6-2=          4
            5*3=         15
            6/8=            .75
```

PeCos evaluated each of the nouns. In displaying them it placed all 4 unevaluated nouns on the left, and vertically aligned them. It placed an equal sign, and then vertically aligned the results.

What does PeCos do if the noun is *a Logical expression?*

```
Display 1+1=3. E/E
        1+1=3=false
```

Again, PeCos placed the unevaluated expression on the left side and then displayed an equal sign and the evaluation of the expression.
Now, what happens if the noun is a *textual expression?*

```
Display "This is text". E/E.
         "This is text"="This is text"
```

Since in a display statement PeCos always places the unevaluated expression on the left of the equals sign, and the evaluated result on the right, display of textual expressions occur as above. Therefore, it is not practical to display text as above. A better way to do it, is by using a form statement or an inform statement, which are described on Page 58.

Rememeber, the nouns in a display statement can be any of the allowable PeCos nouns, (not just expressions). As we go on we will see what **a** display statement does with other nouns.

Set

The set verb is used to assign a value to a variable name.

THE GENERAL FORM OF A SET STATEMENT IS:

Set _______________=______________________ . End sentence

period.

```
     single          value optional if clause
     letter
     variable may
     be subscribed or
     not.
```

The value can be either

1) an expression

2) any of the words-true, false or null.

Example

    Set i=.05*1000

i is the variable name

.05 * 1000 is the expression.

The set statement evaluates the right hand side of the equals sign, and assigns the value to the variable named on the left side. Now when ever i is used, its value will be 50 (ie .05 * 1000)

Now type in:

```
    Display i.
            i=50
```

We used the variable named *i* in a display statement, and PeCos found its value to be 50.

2). What if we give another set statement using *i* as the variable name.

```
    Set i=.05*2000
```

Whenever, a variable name is used, any previous definitions involving that varibale name are lost, and the new definition replaces the old.

```
    Display i. E/E
            i=100
```

3). A set statement may be either direct or indirect. In the direct mode only, the word set and the end sentence period maybe omitted.

```
    i=.05 * 3000 E/E
```
(Note: No word set or end period).

```
    Display i. E/E
             i=150
```

Example	*Assigns the numerical value of 4+6 to the letter a*
a=4+6	
b="sample"	*Assigns the textual value "sample" to the letter b.*
c=1+1=2	*Assigns the logical value of 1+1=2 to the letter c.*

Display a, b, c.

```
        a=10
        b="sample"
        c=true
```

Example

Next let's try to do a program to calculate simple interest.

Remember, interest=rate times amount of deposit

```
                  i= r*d.
```

Let's see how much $1000 at 5% yields

```
d=1000
r=.05
i=r*d
```

Display i.

```
        i=50
```

How about at 6%

```
r=.06
```

Display i.

```
         i=50
```

That doesn't seem right.

What happened was that PeCos only evaluates a set statement when it interprets it. At this time it does the assignment. When PeCos evaluated the set statement for i, r was equal to .05, if we change r, we have to have PeCos do the set statement for i again.

```
i=r*d.                        (r is now .06)

     Display i.
            i=60
```

Example

type in:

```
g=h*3.25
h=???                         This is an error comment. It
                              indicates you used the variable
                              named h, which PeCos does not have
                              a value for. Well, let's give h, h=40
                              a value then try it.
g=h*3.25

Display g.
       g=130
```

Example

```
x=20
x=x+2
```

What does this mean where the variable name (x) is on both sides of the equals sign? Well, when PeCos evaluates a set statement, it first looks only at the right side, does the evaluation using all known values, and then does the assignment to the variable named on the left. In this case it finds x=20, adds 2 (to get 22) and now assigns 22 to x. (remember new definitions of a variable replace old ones).

```
Display x.
        x=22.
```

Example -Setting elements in an array

```
a(1)="Monday"
a(2)="Tuesday"
a(3)="Wednesday"
a(4)="Thursday"
a(5)="Friday"
a(6)="Saturday"
a(7)="Sunday"
Display a(1), a(2), a(3), a(4), a(5), a(6), a(7).
        a(1)="Monday"
        a(2)="Tuesday"
        a(3)="Wednesday"
        a(4)="Thursday"
        a(5)="Friday"
```

```
    a(6)="Saturday" a(7)
    "Sunday"
```

DO

This is a command verb that initiate execution of a program part, or step.

The basic Do statement is:

Do noun.

The noun is either a step, followed by a step number or part followed by a part number. The numbers can be a numeric expression as long as the evaluation of the expression is a legal part or step number.

After completion of what the Do command has ordered, control is returned to the point just after the Do.   If Do was given as a direct command, then control is returned to the keyboard. If Do was given in an indirect (stored) command, then control is returned to the step just after the Do step in the same part, and computation continues.

Example 1.

```
1.1 Display 1.1
1.2 Do part 2.
1.3 Display 1.3.
2.1 Display 2.1.
2.2 Display 2.2
Do part 1.                    Direct command instructs PeCos to
                              find part 1 and start at the
                              first step.
```

THE SEQUENCE OF STEP EXECUTION WILL BE

Step 1.1	
Step 1.2	*Says to do part 2*
Step 2.1	*Part 2 is executed sequentially*
Step 2.2	
Step 1.3	*Control* return *to step 1.3* after *part 2 is executed.*

The results on the screen will be

```
1.1=1.1
2.1=2.1
2.2=2.2

1.3=1.3
```

1.1=1.1	*(Step 1.1)*
2.1=2.1	*(Step 2.1)*
2.2=2.2	*(Step 2.2)*
1.3=1.3	*(Step 1.3)*

Review of Chapter

Let's write a small simple program to see what we have learned. For an employee, given his hourly rate, and hours worked, we would like to calculate:

a. gross pay

b. deductions

c. net pay

1. Let's first pick variable names

h=hours worked

r=hourly rate

g=gross pay

d=deductions

n=net pay

2. What equations do we need?

g=r*h (equation for gross pay).

d=15 percent   gross pay (use 15% for deductions)

n=g-d (net pay is gross pay minus deductions)

3. Lets arbitrarily select part 2, to do the calculation of g, d, n.

2.1 Set g=r*h. *(Grosspay)*

2.2 Set d=.15*g. *(Deductions)*

2.3 Set n=g-d. *(Net pay)*

4. Now lets write part 3 to display the results

3.1 Display ■. *This display■causes an erase of the screen.*

3.2 Display g, d,_,n. *The _ causes a blank line to be displayed.*

Maybe we should also display what r & h are. Well, we can easily insert a step into part 3.

3.15 Display r, h. *Step 3.15 will be done before 3.2. Remember PeCos reorders step numbers into ascending order.*

5. Next let's use part 1 to control the order of the other parts.

1.1 Do part 2.

1.2 Do part 3.

We are ready to run the program, we start by typing in a direct command.

```
Do part 1.
I'm at step 2.1
r=???
```

What does that mean? Something happened at step 2.1, and PeCos can't continue. r=??? means we used a letter r which PeCos doesn't have a value for.

Let's see step 2.1 by using a direct display statement.

```
Display step 2.1
2.1 Set g=r*h.
```

O.K. let's define r&h.

```
Type r=3.25 E/E
     h=40    E/E
now start again
     Do part 1.
             r=3.25
             h=40
             g=130
             d=19.5
             n=110.5
```

Do you want to see what the program looks like--type in Display part 1, part 2, part 3.

All stored steps of those parts will be displayed. Notice how part and step can be used in a display statement. Let's change h, and run the program again.

```
Type h=35
    Do part 1.
```

You should get different results.

Leave this program in PeCos, and see some neat things we can do.

```
f=(3+2)/5. Do

part f.
```

What happened? Did it run our program, starting at part 1? What's f? We set f=(3+2)/5, which=1. So, all we did was say, Do part 1. The part number can be a numerical expression.
Try:

```
Do part (3*3)/9.
```

How about:

```
Do part 4.
I can't find part 4.
```

We asked PeCos to do a part which it doesn't find. This is another of PeCos' error comments.

```
Do step 1.1.
```

Nothing happened? Sure it did, but step 1.1 doesn't cause anything to be displayed. Look at step 1.1 by typing:

```
Display step 1.1.
```

Let's see how we change the program, instead of deductions of 15% of gross, let's also deduct $2.00 miscellaneous.

Display part 2. *(Part 2 does the calculations)*

```
2.1 Set
g=r*h.
2.2 Set
d=.15*9
2.3 Set
n=g-d.
```

Add step 2.25 as follows

```
2.25 Set d=d+2.
```

This takes the present value of d, adds 2, and then sets this value equal to d.

Also, notice that this step occurs between 2.2, and 2.3 PeCos reorders steps into ascending order regardless of the order of entry. Now run the program again.

d & n should have changed.

Next let's change deductions for taxes, to 17% of gross simple-rewrite step 2.2

```
2.2 Set d=.17 *g.
```

If you use a step number that is previously used, the new statement replaces the old, now run the program again. Let's move on to the next chapter, to learn more of PeCos intelligence.

Chapter 7.

FORMS

A form statement, allows you to describe how you want the output to appear. The format of writing a form statement is as follows:

Form n: *(Touch Enter/Execute)*

Up to 80 characters of text, numbers or fields.

(Then touch Enter/Execute, again).

First, notice that we have touched Enter/Execute twice. Once, after we type in the word, Form, a number, and a colon. Then we touch Enter/Execute again, after we type in the description of the form. This is the only statement in PeCos where you can and must touch E/E twice.

n, is a number, numeric variable, or numeric expression and must be an integer between 1 and 999999999. (a 9 digit maximum).

FORMS WITH NO FIELDS

Forms with no fields, are useful in displaying headings, messages, and the like. In this mode, the 80 characters of the form definition will be displayed upon request exactly as they were keyed-in. The command is simply:

Display form n.

Example 1.

Form 2: *(Touch Enter/Execute)*

This is a heading or message statement.

No quotes are required. *(Touch Enter/Execute).*

The above defines form 2. *(Next type in, Display form 2).* Display form 2.

(Result on screen is:)

This is a heading or message statement

No quotes are required.

Example 2.

Form 3: *(Touch Enter/Execute)*

PAYROLL *(Touch Enter/Execute)*

Form 4: *(Touch Enter Execute)*

Please enter the employees name, and

number when requested, then the *(Touch Enter/Execute)*

Form 5: *(Touch Enter/Execute)*

Hourly rate, and number of hours worked. *(Touch Enter/Execute)*

10.1 Erase *(This step erases the screen)*

10.2 Display form 3, form 4, form 5. *(Note: That forms are like any other noun in a display statement).*

Do part 10.
(Result on screen is :) PAYROLL

Please enter the employees name, and

number when requested, then enter the

hourly rate, and the number of hours worked.

Let's try the following:

Display form 22.

I can't find form 22.

As we have seen previously, if you ask PeCos to display something non-existent, PeCos tells you that it can not find what you are looking for.

EXAMPLE

(Type in the *following:)*

Form 10:

Hi! My name is PeCos. I'm very new
in this world but, I am very smart.

Form 11:

I am giving you a sample of how I can
handle textual statements.

Form 12:

Input any characters in a form
statement.

Form 13:

I'll reproduce them exactly as you tell
me.

In the above, each form contains text, which we would like to use to create a paragraph. Just use the form numbers in a display statement, such as Display form 10, form 11, form 12, form 13.

```
Display form 10, form 11, form 12, form 13.
```

(Result on screen is:)

```
Hi: My name is PeCos.   I'm very
new
in this world but, I am very
smart.
I am giving you a sample of how I
can handle textual statements.
Input any characters in a form
statement.
I'll reproduce them exactly as you
tell me.
```

We can modify this by putting in some blank lines in the display statement.

```
Display ■ , form 10,_, form 11,_, form 12,_, form 13,_.
```

(Result on screen is:)

```
Hi'.  My name is PeCos.  I'm very
new
in this world but, I am very
smart.

I am giving you a sample of how I
can handle textual statements.

Input any characters in a form
statement.

I'll reproduce them exactly as you
tell me.
```

(Next type in:)

a=10

Display form a.   *(Pecos will display form 10).*

Of course, *the number of a form can be a* numeric expression.

(Now try the *following:)*

5.1 Set x=10

5.2 Display form x.   *(PeCos will display form 10).*

5.3 Set x=x+1.

5.4 Display form x.

Do part 5.   *(Did PeCos display forms 10, and 11?*
it should have).

FORMS WITH FIELDS

PeCos allows you to specify a format for how values, values of variables, or values of expressions should be displayed. Normally when we display any of these, the format is fixed. *(Review the display statement*). We can use an in form modifier with a display statement, which will tell PeCos how to display the values in the display statement. This is done first by defining a form statement with *Fields*,(the fields show where and how the values are to be placed).

A field consists of:

a. a *series of underscores (_ _ _ _ _)*

b. a series of decimal points (. . . . . . .)
(a minimum of 5 decimal points must be used).
c. a series of underscores with one decimal point (_ _._ _)

A form with fields is defined by PeCos exactly like forms without fields.

Let's try one

Form 20: *(Touch* Enter/Execute)

Name: _ _ _ _ _ _ _ _ _ _ *(Touch Enter/Execute)*

Just like any other form.

O.K. Now the underscores (10 of them) are known as a field.

Form 20, can be used as we did previous forms.

Display form 20:
Name:_ _ _ _ _ _ _ _ _ _

and you can see it displayed exactly as it was entered.

or we can use it in an *in form clause,* of a display statement. In this case, the value on the display statement will be filled into the field of the form.

EXAMPLE

Display "John Smith" in form 20.

(The result is)

Name: John Smith

The value of the textual noun in the display statement was filled into the field defined in Form 20.

The general rule of an in form statement is: where a display in form statement is used, values will be filled into the fields in the order specified, and rounded or truncated when necessary.

Let's see some examples.

Form 20:

Name: _ _ _ _ _ _ _ _ _ _ Age: _ _ _

Note: we *have rewritten form 20 with* the new
definition re-placing the old.
Display "John Smith", 29 in form 20.

(The result is)

Name: John Smith Age: 29

The value of the first noun (John Smith), is filled in to the first field of Form 20, and the value of the 2nd noun (29) is filled into the 2nd field of Form 20

Some other points to note are that the first field has 10 underscores. This means there is room for up to 10 characters to be filled in. The 2nd field has 3 underscores, which allows up to 3 characters. What happens if the value of a noun has more characters than allowed for in the field.

Display "John R. Smith.", 29 in form 20.

Name: John R. 5m   Age 29

Only the first 10 characters (starting from the left) were filled into the field of the form. We only allowed in the form definition for 10 characters to be displayed, and that's all PeCos could do. Always remember to leave ample room in a field. Perhaps, a better way to define form 20 is:

Form 20:

Name: _

Age: _ _ _

Now try to display "John R. Smith".

What happens, if we don't display the same number of values as there are fields in a form statement?

Form 25:

First name_ _ _ _ _ _ _   Second name _ _ _ _ _ _

Display "John" in form 25

First name John   Second name _ _ _ _ _ _

We only had one value (John), displayed in form with 2 fields. Notice, the 2nd field, (as well, as the rest of the characters of the form) are displayed as they were entered. Fewer values than numbers of fields in a form is O.K. How about:

Display "John", "R", "Smith" in form 25.

(PeCos responds with:)

I have too many values for a form.

You can not try to display more values than there are fields defined.

Now try

```
Set a="John"

Set b="Smith"

Display a, b, in form 25.

First name John              Second name Smith
```

Notice that it is the value of the variable or expression that is placed in the field, and not the name of the variable.

Before, we mentioned 3 ways to define a field, (underscores, decimal points, or underscores with 1 decimal point), up to now we have used only underscores, so what are the other modes for?

For displaying textual or logical values in form, the method of defining the fields in a form doesn't matter. You can use any of the 3 methods. It is only with displaying of numerical values in a field that the mode of field definition will effect the display.

EXAMPLE

```
Form 40

1st value _ _ _ _ _           2nd value _ _ _._ _
3rd value . . . . .
```

The first field has 6 underscores, the second field has 3 underscores, a decimal point. and then 2 under-scores, and the third has 6 decimal points.

```
Set a="Fred"

Display a, a, a in form 40.

1st value Fred               2nd value Fred

3rd value Fred
```

All three fields treated the value of the variable a, in the same way. So for a textual variable we can define a field in any of three *ways.*

Next let's try a *logical variable.*

Set b=2+3=5. *(b will have a* value *of* true).

Display b, b, b in form 40

1st value true 2nd value true

3rd value true

Again, all three fields treated the value of the variable a in the same way.

For numerical values the rules are as follows:

1. If only underscores, then only the integer portion (the digits to the left of the decimal), will be displayed. The value will be rounded if there is a decimal portion.

2. Underscores, and one decimal point-Both an integer and decimal portion will be displayed. The number of underscores to the right of the decimal point tells PeCos how many decimal digits to display.

3. If decimal points are used, numbers will be displayed in in scientific notation-a sign of mantissa, a mantissa, 2 digits exponent of 10, and a sign exponent. *Note: If you use dots to define a field, the minimum number of dots is 5, regardless of the type of value to use that field.*

Example

(using Form 40 as previously defined)

Set c=12.345.

Display c, c, c in form 40.

1st value 12 2nd value 12.34

3rd value 1.01

Well, we got three different types of values. Let's see what they mean, and what happened.

1. The value we tried to display was 12.345.

2. In the first field we had 6 underscores. Pecos placed only the integer portion of 12.345 in this field. So when a numeric value is displayed in a field with only underscores, then only the integer portion of the number will be displayed.

3. In the 2nd field, we had 3 underscores, a decimal point, then 2 underscores. The position of the decimal point told Pecos to display 2 digits to the right of the decimal point, and 3 digits to the left. Also Pecos will round the decimal por-tion if the value has more decimal places than the field says to display.

4. In the 3rd field, we had 6 decimal points, and we got a 1.01. This is a form of scientific notation the mantissa is 1, and the exponent of 10 is 01. *(ie, 1* X *10^{01}).*

When using a field of decimal points to display a numeric value, Pecos will use up the decimal points in the following priorities:

a. 2 decimal points for exponent

b. 1 decimal point for sign of exponent

c 1 decimal point for sign of mantissa

d. The rest of the decimal points, specify how many digits of the mantissa (including decimal point) will be displayed.

Let's do some examples of numeric values in form.

Form 41:

_ _ _ _ _

x=12.6

Display x in form 41.

13

Since we asked for a number with a decimal piace, to be placed in a field which doesn't allow for decimal places, PeCos rounded the values. Pecos rounds up if the next digit is 5 or more rounds down if the next digit is less than 5.

Form 42:

. . . .

x=123.4

First note that x can be rewritten in scientific notation as $x=1.234x10^2$

Display x in form 42.

1.23 02

The 8 decimal points were used as follows

2 for exponent

2 for sign of exponent and mantissa

4 for mantissa (including decimal point).

Chapter Review

Example

Let's do a simple interest program using forms

First let's define the forms

Form 1: *(Will* be used as a heading)

```
                 SIMPLE INTEREST CALCULATION
```

Form 2:

```
Deposit                        Rate           Interest


$ _ _ _ _ _ . _ _            _ _ . _ _%     $_ _ _ _ . _ _
```

Form 2 is used to display the deposit, rate, and interest paid. It has 3 field, (1 for each value), with decimal places specified.

Now for the program

```
1.1 Set i=r/100*d.
1.2 Display ■ , form 1,_.
1.3 Display d, r, i in form 2.
```

Step 1.1 does the calculation.

Step 1.2 does an erase, displays form 1, then a blank line

Step 1.3 displays the values of d(deposit), r(rate), and i(interest) as specified by form 2.

```
r=6
d=1000
Do part 1.
                 SIMPLE INTEREST CALCULATION

Deposit                        Rate           Interest
$1000.00                       6.00%          $60.00
```

Chapter 8

USING ARRAYS

As previously mentioned, letters can be used as names of subscripted variables. A group of subscripted variables with the same name is known as an *array.* The value of each element of an *array* is established, in the same *way* as the value of a nonscripted variable, (ie, by a *set*, *read, or demand statement*).

The Rules of Arrays:

1. The name of the arrays is a single upper or lower case letter.

2. An array can have from 1 to 10 subscripts (indices).

3. Each subscript can be in the range of -999 to +999. They must be integers (no fractions allowed).

4. The subscript itself, can be a variable name, or numeric expression, but the value of this variable must be an integer between -999 to +999.

4a. A subscript is also called an index.

5. An array can have only one dimension at a time. That is you can not use a letter to name an array with 3 subscripts and then later use the same letter to name an array with 2 subscripts.

6. All members of an array must have values that are of the same type. That is, the value of all elements of a named array must be either numerical, logical, or textual.

Example

1). Set x(1,1)=3*4+2. *Defines array named x, all members of the array, must have 2 subscripts This particular member has subscripts 1.1. The value of the member 1,1 of array x, is the value of the expression 3*4+2, (a number).*

```
2).  x=10.                        Sets elements of array a, equal to
    Set a(1,1)=x.                 arithmetical expressions.

    Set a(1,2)=x+2.

    Set a(2,1)=x+6.

    Set b(2,2)="sample 1".       Sets elements of array b,
                                 equal to textual value.

    Set b(2,3)="sample 2".

     Set c(1,1)=true.            Set element of array c, equal to
                                 logical expressions.




3).  Using the set statements of
    example 2.

    Display a(1,1), a(1,2), a(2,1).


    a(1,1)=10

    a(1,2)=12

    a(2,1)=16

4). Set m(1)=2+3+4.               Set elements of array mequal to
                                  numerical expressions.
    Set m(2)=2*3*4.

    Set x=1.

    Display m(x), m(x+1).         A subscript can be represented by a
                                  numerical variable, or numerical
            m(x)=9.               expression.
          m(x+1)=60.

    Set m(12)=6+7

    Set m(m(12*x) )=2.            This set m(13)=2.

    Display m(13).

            m(13)=2.
```

ALL

The word All, is *a keyword* of PeCos, and can be used with the following nouns to create noun phrases.

1. Parts
2. Steps
3. Forms
4. Values

These noun phrases, can be used in a *display, delete, file* or *print* statement.

The command verb will act upon the entire noun phrase.
If the word **all**, is used by itself, as a noun, in any of the above statements, PeCos assumes it to mean **all parts**, and **all forms** only.

Example
Type in the *following:*

```
1.1 Set a=4.
1.2 Set b=a+c.
1.15 Set c=3.
2.1 Display a, c, b, in form 1.
Form 1:
a=_ _ _. _ _        c=_ _ _ _ _._ _         b=_ _ _._ _


Form 2:
                          This is a sample.
```

Now type in

```
Display all forms.

Form l:

a=_ _ _ _._ _                c=_ _ _ _._ _            b=_ _ _._ _


Form 2:

                       This is a sample
```

Note: The form number, is displayed with the form in the above example. If you had only said:

```
Display form 1, form 2.
```

(The result would have been:)

```
a=_ _ _ _._ _                c=_ _ _ _._ _            b=_ _ _._ _


                       This is a sample
```

```
Display all parts.

1.1 Set a=4.

1.15 Set c=3.

1.2 Set b=a+c.
```

PeCos reorders step numbers into the correct sequence regardless of how they were entered.

```
2.1 Display a, c, b,
```

There is a blank in between the end of part 1, and part 2. Whenever PeCos is displaying all parts it puts these blanks in to improve readability.

Delete all forms.

This will erase *all forms from* the PeCos *memory.*

Display all forms.

Two blank lines are output to show there are no *forms stored.*

Display all.

PeCos responds *by displaying* all parts, *and forms.*

1.1 Set a=4.

1.15 Set c=3.

1.2 Set b=a+c.

2.1 Display a, c, b, in form 1.

(Note: *PeCos always places blank lines* between parts, and *forms, two blank* lines are output *to show* there are no *forms).*

Do part 1.

Part *1 will set up 3 variables.*

TO

This is **stored command** only. It will change the order of step execution. Unlike the **Do Statement**, control will not return to the step following the **To Statement**, but continues as directed at the step reached from the **To Statement**. **To** can send the program forward or backward, or to itself. The General Form is:

To noun "if clause".

noun is step number or part number

optional if clause

Example:

```
1.1 Display 1.1.
1.2 To part 2.
1.3 Display 1.3.
2.1 Display 2.1.
2.2 Display 2.2.
```

The sequence *of* steps *for the above,* when *a Do part 1, is given will be:*

```
Step 1.1
Step 1.2                    Directs to step 2.1
Step 2.1
Step 2.2
```

Step 1.3 is never returned to after part 2 is completed, return is given to the step giving the last Do command, (Which was a direct command, Do part 1).

Example

```
To step 1.1.
```

Don't give this command directly. Remember this can only be a stored command.

Example

Let's do the following:

1. Set up an array with 3 names of people.
2. Set up a second array with ages of these people.
3. We will then display (using a form statement) the names, and the ages.

First, let's set up the data, array a, will be the names, and array b*,* will contain their ages.

```
a(1)="Joe Smith"
a(2)="John Jones"
a(3)="Mary Doe"
b(1)=28
b(2)=37
b(3)=29
```

Now let's set up the form to display the names and ages.

```
Form 1:
Name                          Age
Form 2:
_ _ _ _ _ _ _ _ _ _ _ _ _    _ _
```

Now, for the program.

```
1.1 Set x=1.
1.15 Display ■,_,form 1,_.
1.2 Display a(x), b(x)   in form 2.
1.3 Set x=x+1.
1.35 Display _.
1.4 To step 1.2.
```

The program, uses the numeric variable **x**, as the index of arrays **a**, and **b**. We start off with x=1, (the first member of arrays **a**, and **b**). Step 1.2 displays the values of a(x), b(x), using Form 2. Since we just set x=1, it will be the values of a(1), and b(1). Step 1.3, changes the value of x, to x+1. (2). Then step 1.4 sends the program to 1.2 which now will display the values of a(2), and b(2), using Form 2.

Let's try it,

```
Do part 1.
Name                             Age
     Joe Smith                       28
     John Jones                      37
     Mary Doe                        29

I'm at step 1.2.
a(4)=???
```

The program did what is called a **loop**. At step 1.3. it set x=x+1. Then at 1.4 it sent it back to step 1.2. When x reached 4, and PeCos was asked to display a(4), it found there was no a(4). The program went O.K., until we reached x=4. We will learn various ways of getting out of loops, before it's too late.

FOR

The word **For,** can be used only with **a Do Statement.**

It gives the group of values for which a Do order is to be executed.

1). The format is:

Do noun for N=a(b)c.

The noun is a part or step numebr.

a). N is a variable (numerical type *only).*

*b). a(b)c specifies the range values of **N**, for which the part or step is to be done. This is like repeatedly giving a **Set Statement** for **N**, and then giving a **Do Statement** after each new value of **N** is set. **a(b)c**, are numerical variables, or expressions. They are interpreted as **a** is the first value, **b** is the increment or decrement, and **c** is the end value.*

That is:

1. The first value for **N**, is **a**.
2. The next value a+b, (increment last value of N by b).
3. The next value is a+b+b (take last value for **N**, and increment by b).
4. Continue but do not go beyond **c**.
5. **c** is the last value for **N**.

Example

1.1 Display x.

Do part 1 for x=1(2)11.

The above statement says to do part 1, for the following values of x; x=1 is first value, then Do part 1 again, but with x=1+2=3.

Then do part 1 with x=3+2.

Continue until the end limit, (11) is reached.

The result is:

```
   x=1
   x=3
   x=5
   x=7
   x=9
   x=11
```

Example

Let's try the example on page 74 and 75, with a for clause,

Input array **a**, array **b**, as before. Also form 1, 2. Let's re-write part 1.

```
1.2 Display a(x), b(x) in form 2.
Display ■ , form 1.
Do part 1 for x=1(1)3.
```

```
Name                       Age
Joe Smith                      28
John Jones                     37
Mary Doe                       29
```

No error at the end this time, and we got our results.

Example-2

a, b, c can be numerical expressions

1.1 Display x.

Set y=1.

Set z=7.

Do step 1.1 for x=y(2*y)z.

```
                X=1          (First value is y which =1)
                X=3          (increment is 2*y which -2)
                x=5
                x=7          (last value is z which equals 7)
```

Example-3

The last value of the range is always done exactly.

1.1 Display x.

```
                X=1             first value
                x=3            Last value of x plus increment
                x=5           last value of x plus increment
                x=7
                x=9          last value plus increment
                X=10          last specified value always done
```

4. The last value can be smaller than the first, in which case you can use a decrement value.

Example:

1.4 Display x.

Do step 1.1 for x=5(-1)-l. *(Note,* the increment *is -1).*

x=5 *First value.*

x=4 *Next value is last value -1.*

x=3

x=2

x=1

x=0

x=-1 *Final value.*

5. There can also be multiple ranges by using commas in:

Do noun for N=a(b)c(d) e, f, g.
This is interpreted as:

N=a, then N=a+b, a+2b etc till c is met.

Then c+d, c+2d, c+3d, etc till e is met, then f, then g.

Note: The end value, (c, and e in the above) is always met exactly.

Example:

1.1 Display i.

Do part 1 for i=2(3)10, 15, 20.

i=2

i=5

i=8

i=10 *End points hit exactly.*

i=15

i=20

EXAMPLE

```
1.1 Display x.

1.2 Set x=10.

1.3 Display x*x.

1.4 Display _.

     Do part 1 for x=1(1)3.

             x=1

             x*x=100

             x=2

             x*x=100

             x=3

             x*x=100
```

Note that values of x, specified by the ***for clause*** *are retained and used in successive iteration, even though x, has a new value at step 1.2.*

```
Display x.

             X=10
```

EXAMPLE

Set a=3.

Set b=4.

1.1 Display i.

Do part 1 for i=a(a+2), a+b, (b+1), a+15, 50, .001

i=3	*value of i=a*
i=5	*Value of i=a+2*
i=7	*Value of i=a+b*
i=12	*Value of i=a+b+b+1*
i=17	*Value of last value + (b+1)*
i=18	*Value of i=a+15*
i=50	*value of i=50*
i=.001	*Value of i=.001*

PRINT

The verb print, is used to output through the RS232C connector to an optional printer.

Print, is used exactly like the display verb, except the output goes to a printer instead of the CRT.

The General Form Of A Print Statement Is:

```
Print   ________    ______________________

        Nouns       Modifiers or If
```

Clause.

Some examples are:

Print 1+2.	*Single nouns.*
Print 1+2, 3-4, 6*8.	*Multiple nouns.*
Print a, b*c.	*Variables.*
Print part 1, step 2.2.	*Parts, steps.*
Print all parts.	*All parts.*
Print all forms.	*All forms.*
Print all.	*All parts, all forms.*
Print a if b=6.	*With an if clause.*
Print all parts if timer -a=6.	
Print a, b, c in form 1.	*With an in form modifier.*

NOTE: When installing a printer, make sure the baud rate switch on the rear of PeCos is set to the same baud rate that the printer is operating at.

DEMAND

The **demand statement**, is used to ask for an input for a variable to be entered from the keyboard. This is an indirect command only. There can be an optional **if clause.**

The general form of the statement is:

Demand Noun as "Text" "If clause"

1. The noun is a variable, (subscripted or not). The type of variable, textual, or logical) must have been previously defined. Then the input from the keyboard will be treated as this type. This means that if the variable is numerical, and you input a text or logical variable, an error will occur or the input will be treated incorrectly.
2. The as "text" is required. PeCos will output to the display upon executing the demand statement

   Text=

   At this time, an entry from the keyboard is made, and touch Enter/Execute to indicate when the entry is completed. (No period is necessary).

Example

```
1.1 Demand d as "d".
Do part 1.
d=
```

PeCos has reached step 1.1, and indicated, that it is waiting for an input. To answer PeCos, just key-in a value for **d,** and then touch E/E, (no period necessary).

```
d=4 E/E
```

Now type in:

```
Display d.

        d=4
```

```
1.1 Demand d as "Amount of Deposit".
Do part 1.
Amount of deposit=
```

Instead of displaying the symbol **d**, PeCos displayed the text we wrote in the demand statement.

Type in 1000, and Touch E/E. Now type in Display d.

```
     d=1000
```

Let's try a payroll calculation.

Example

1.1 Demand h as "Hours Worked".

1.2 Demand r as "Hourly Rate".

1.3 Set p=r*h. *(Uses input for a calculation)*

1.4 Display r, h, p in form 1.

Form 1:

```
Rate$_ _ _ ._ _       Hours Worked _ _ _    Grosspay$ _ _ _ _ _ ._ _
```

Do part 1.

Hours Worked=40 *(Input 40, touch, Enter/Execute)*

Hourly Rate= 5.25 *(Input 5.25, touch Enter/Execute)*

```
Rate $ 5.25           Hours Worked 40       Grosspay$ 210.00
```

The Demand statement and type of variable.

A demand statement can not set the type of variable. PeCos is already expecting the input to be of a certain type, (numerical, textual or logical), and will treat the in put value as such.

As an example, If you demand a value for the variable **a**, and PeCos is expecting it to be textual, then no matter what you input, (textual logical or numeric), the value will be treated as textual.

The type of variable is defined to PeCos in one way :

1. With a **Set** or **Read** statement. When a variable is used in a set or read statement, it's type is determined by the type of value it is **set** to, or **read** from

Example

```
      Set a=2

          a, is a numeric type of variable.

      Set a="Text".

          a, is now a textual type.

Read a. (from tape)a will take on the type of variable
                    that is read.
```

Since a demand statement, must know the type of value before the input, it is necessary to define the type before **a** demand statement.

If you don't define the type, PeCos will assume, the variable to be numerical, if the variable was not already used in a set or read statement.

Example

1.1 Set="text"

Dummy statement defines variable a to be textual.

1.2 Demand a as "a".

1.3 Display a.

Do part 1.

a=

Let's act dumb, and input a numeric value 6.

6 E/E.

    a="6".

6 is displayed with quotes, indicating it is a textual value.

2.1 Demand c as "c".

2.2 Display c.

Do part 2.

c=

Let's act dumb again, and input something as text.

John E/E.

J=???

PeCos responds with an error comment for an undefined variable.

It though we input a numeric value named J, (first letter of John), and did not find a value for J.

Add step 2.05

2.05 Set="c".

Then try part 2.

IF CLAUSE

The word **if**, may be used to qualify any command verb. It specifies the conditions under which the command verb will be executed. These conditions, may contain any logical expression which can be evaluated to be true or false.
In evaluation of a command with an **if clause**, PeCos first evaluates the condition. If the condition is not met, the command is not interpreted, or processed at all.
(Thus, the command could be in error, and not detected unless the condition is met).

The General Form is:

<u>Command</u>   <u>Noun</u> <u>if</u> <u>Condition</u>

The command can be any of the command verbs.

The noun can be any allowable nouns, (or more than 1 noun if that type of command permits it).

Example

```
Set x=1.
Set y=4.
Display x if y<35.          y< 35 is a logical expression)
        X=1               Condition met so perform command verb
                          Display

Display x if x=3.          (Condition not met).
```

Examples-continued

```
Set y=35 if x < 40.
Display y.
    y=35
Example
Set x=1.
Set y=35.
1.1 Display x.
Do part 1 if y > 10↑6.       (Condition fails)

Display garbage if y=0.   (Condition fails, so command is
                           not interpreted, Note: that
                           display garbage is an incorrect
                           statement).
```

REVIEW

Let's do the simple interest problem

1. Input the rate, and deposit from the keyboard.
2. Keep inputting various rates, and deposits till we enter that we are finished.

1st the forms for the output

Form 1:

Simple Interest

Form 2:

Deposits_ _ _ _ _._ _ Rate_ _._ _%

Interest_ _ _ _ ._ _

Now we need parts to do the following:

1). a control part (part 1)

2) to do the input (part 2)

3) to do the calculation(part 3)

4) to do the output (part 4)

```
2.1 Demand d as "What is the deposit".
2.2 Demand r as "What is the interest rate".


3.1 Set i=r/100*d.
4.1 Display d, r, i in form 2.
```

Part 1 will also do any initialization we need, and also check if there are any ::more inputs.

```
1.1 * simple interest calculation
1.2 Erase.
1.25 Display form 1.
1.3 Do part 2.
1.35 Do part 3.
1.4 Do part 4.
1.5 Set z="z".
1.55*Step 1.5 set z to be a textual variable.
1.6 Demand z as "anymore entries",
1.65 To step 1.3 if z ? "no",
```

Step 1.6 requests from the keyboard, if there are any more entries.

Step 1.65 says to go to step 1.3, if the value of z is not equal to "no".

If you input anything other than no from the keyboard, then step 1.65 will be done, (condition is true) and we will go to step 1.3

Let's try it.

Do part 1.

```
What is the deposit=1000
What is the interest rate=5.00
Deposit $1000.00                Rate 5.00%
        Interest $50.
Anymore=yes               (Note: No quotes needed)
What is the deposit=
```

The program went back to step 1.3 since we didn't enter no.

now enter

2000 E/E

What is the interest rate=5 E/E

PeCos gives the output for the new d, and r then it says:

Anymore=

Just touch E/E, and see what happens.

It continues with the program since touching E/E entered nothing for the variable z.

Next time it says, Anymore=

Enter no E/E.

It's finished.

CONDITIONAL EXPRESSIONS

A conditional expression is a **noun.**

The simple notation for a conditional expression is:

[A:B;C] *The brackets (or parenthesis)are a must).*

This is interpreted as:

1. A is some expression that can be interpeted as being true or false.

   If it is true, then use the value after the colon, and if false use the value after the semicolon. The values B, C must be of an appropriate type, for the variable in the sentence.

2. A conditional expression can be used anywhere that a noun can be used.

Example

Set a=[2=3:5;6].

The expression 2=3, is evaluated to be false, so the value

used will be after the semicolon, (6) and a=6 is the result.

3. The expression can be simple or complex.

Example

Set a=2

Set b=4

Set c=true

Display [a>l:b+a;false], [b>a:c;"cat"].

[b+a] =	6	*The evaluation of the first expression is true, so use b+a.*
[c] =	true	*The evaluation of the second expression is true, so use c.*

Example

1.1 Set a="Smith".

1.2 Set b=[a="Jones":"John";"Joe"].

The conditional expression in the **set b statement** check to see if **a**="Jones". If it is Jones, then Set **b**=John. If it is not "Jones", Then **set** b="Joe".

Do part 1.

try Display b.

    b="Joe".

Now change a.

1.1 Set a="Jones".

Do part 1.

Display b.

    b="John".

4. The values to be used, can themselves be expressions when the form becomes:

   [a:b;c:d;e]

The above is interpreted as if **a**, is true, use **b**, if false then evaluate c. If **c** is true use **d**, if false use **e**.

Example

Display [1=2:10;12:20;30]. *(the first false value to be used is itself an expression to be evaluated).*

    [20]=20

Example:

```
1.1 Do part [a<0:2;a=0:3;a>0:4].

2.1 Display 2.1.

3.1 Display 3.1.

4.1 Display 4.1.


Set a=0

Do part 1.
```

Evaluation of expression in step 1.1 is 3.

```
   [3.1]=3.1

Set a=-6.

Do part 1.
```

Evaluation of expression in step 1.1 is 2.

```
   [2.1]=2.1

Set a=2.

Do part 1.
```

Evaluation of expression in step *1.1 is 4.*

```
   [4.1]=4.1
```

Example *(Type* in the *following).*

```
1.01 Erase.

1.02 Set a="2".

1.1 Demand a as "Do you like me?".

1.2 Display form [a="yes":1;2].


Form 1:
                         Well I like you too!
Form 2:
That's O.K. By the time you finish read-
ing this book, you'll change your mind.
```

Example-continued

(Type in:)

```
Do part 1.                    (See the result, by answering
                               yes or no).
```

DELETE

The verb delete followed by one, or more nouns, (separated by commas), may be used directly or indirectly and will cause these nouns to be deleted from PeCos memory.

The nouns can be a:

1. A part number
2. A step number
3. A form number
4. All (will delete all steps, parts, and forms).
5. A variable name
6. An array name

Example

```
1.1 Display 6+2.
1.12 Set r=fp(291*4).
                                   Part 1.
1.3 Display "I'm at step 1.3".
2.1 Set g=t?(100*ip(r)).           Part 2.
2.2 Display g.


Form 1:                            Form

_ _ _._ _     _ _  _  . . . .
```

Examples-continued

5=22/7 ***variables***

a(1.1)=4 *arrays*

a(1,2)=3

Delete step 1.2, part 2, form 1, S, a.

Display all.

1.1 Display 6+2.

1.3 Display "I'm at step 1.3".

4.1 Delete all. *Delete may be used*

indirectly Do part 4.

Display all. *Blank lines follow to*
indicate nothing stored.

DONE,

Done is a command verb, which causes PeCos to stop execution of a part. It is equivalent to reaching the end of that part, (where there is an **implied done statement).** Control returns to the step following the command that initiated the part. (If it was a direct command, then control returns to the keyboard), (if it was an indirect command then control returns to the next step following that command).

EXAMPLE

```
1.1 Set b=0.

1.2 Set a=1.

1.3 Do part 2.

1.4 Done if b=2.          Will stop all programs, and return
                          control to keyboard if b=2.
1.5 To step 1.2.


2.05 Set b=b+1.

2.1 Display a, b.

2.2 Done if a=3.

2.3 Set a=a+1.

2.4 To step 2.1.


Do part 1.

a=1

b=1

a=2

b=1

a=3

b=1           Step 2.2 is performed so return to part 1.

a=1

b=2

a=2

b=2

a=3

b=2           Step 1.4 is performed.
```

STOP

When a **stop command** is performed PeCos returns control to the keyboard, and displays a message stopped at step N.   Stop is only an indirect command, and can have an If clause. It is very useful to insert a stop statement when trying to **debug** a program.

EXAMPLE

```
1.1 Display 1.1.
1.2 Do part 2.
1.3 Display 1.3.

2.1 Display 2.1.
2.2 Stop.
Do part 1.
1.1 =1.1
2.1=2.1

Stopped by step 2.2.
```

Note: *Part 2 never returned control* to *part 1, the do execution* that *was in process is not cancelled out* by a stop *statement.*

GO

This command verb continues computation of a program after an interruption. It will continue as if the interruption had not occurred. The interruption could have been manual, (by pressing the interrupt key) or automatic, (such as a stop statement, or an error message).

From when the interruption occurred till **Go** is entered you can modify the program (add or delete), input variables etc.

Example

```
1.1 Set a=d+2.

1.2 Display a.


Do part 1.

I'm at step 1.1            Undefined variable so program
                           halts with error message
d=???

Set d=4.

Go.                         Go sends control to next
                            incompleted step.   (step 1.1)
a=6                         was not completed due to an
                            error being found.
```

EXAMPLE

```
1.1 * This program gives the cube of a number.

1.2 Erase.

1.3 Demand a as "What number do you want cubed."

1.4 Display a, a*a*a in form 1.

1.5 Stop.

1.6 To step 1.2.
```

```
Form 1:

The cube of _ _ _ _ _ _   is _ _ _ _ _ _
```

```
Do part 1.

What number do you want cubed= 6    (key in and enter 6)

The cube of 6 is 216

Stopped by step 1.5.

Go.                          (Will continue after stop command).

What number do you want cubed=
```

SIZE

You can use the word size as a noun. Size is the amount of storage space left in PeCos. Memory for storing steps, parts, forms, arrays, variables, and formulas. Upon system initialization size=1864.

EXAMPLE

```
Power Up
Display size.
        Size=1864
1.1 Set i=-999.
1.2 Set a(i)=i.
1.3 Display i, size.
1.4 Set i=i+i.
1.5 To step 1.2.
    Do part 1.
       i=-999
    size=1863
       i=-998
    size=1862
```

You can see that as more elements are added to the array **a**, size is decreasing. When size reaches 9, PeCos will automatically stop and display:

I ran out of space.

When this occurs, you can recover some of the memory space by:

a. Resetting the system -this will clean out all memory, and recover all space.

b. Use a delete command -delete some steps, parts, variables, arrays,

c. Cancel-Will clear out space used for storing re-turn locations from incomplete Do statements.

CANCEL

The verb cancel is used to clear memory of all return locations for any outstanding do statements, interrupts or stops.

EXAMPLE

```
1.1 Display 1+1, size.
1.2 Do part 1.
```

This will cause a jump to start of Part 1., (step 1.1) it will keep repeating this, and you can see the size decreasing. Eventually the size will be 9, and PeCos will stop with a message.

```
Do part 1.

1+1=2

size=1851

1+1=2.

size 1848
    .
    .
    .
1+1=2

size=9.
```

Example-continued

```
I'm at step 1.1.

I ran out of space.

Display size.

       size=9

Cancel.                    Now key in cancel.

Display size.              All spaces used to store return
                           addresses are cleared but programs
       size=1856           are retained.


Display all.

1.1 Display 1+1, size.

1.2 Do part 1.
```

Chapter 10

INTERNAL FUNCTIONS

PeCos has several functions internally which can be used:

The general Form is:

Function(x)

1. "Function", is the particular symbol for the function and x is the variable.

2. There must be no space between the function name and the left parenthesis or bracket.

PeCos computes the true value rounded to nine significant digits;in most cases care is taken to hit certain ***magic values (Such as sin p/6)*** *Where p= ?*

3. Functions are nouns of course, and can be used anywhere a noun is allowed.

MATHEMATICAL FUNCTIONS

Function	***Symbol***	***Conditions***
Absolute value	| |	
exponential (e^x)	*exp(x)*	$e^x<10^{100}$
Natural Log	*log(x)*	*x>0*
Square Root	*sqrt(x)*	*x=0*
Trigonometric sine	*sin(x)*	*|x| 100-radian measurement*
Trigonometric cosine	*cos(x)*	*|x| 100-radian measurement*
Argument	*arg(y,x)*	

NOTE:

Tan^{-1}(x)=arg(l,x)

Sin^{-1}x=arg(sgrt(l-x?2),x)

COS^{-l}x=arg(x,sgrt(l-x?2)

EXAMPLES

```
Display sqrt(2), log(10), exp(1), sin(3.14/15).

    sgrt(2)=1.41421356
    log(10)=2.30258509
    exp(1)=2.71827182
    sin(3.14/15)=9.26536*10?(-5)     (3.14/15 is in radians)
```

Note: a good approximation of ? is 355/113.

EXAMPLE

```
Display exp(log(13.1)).

        exp(log(13.1))=13.1
```

Example

```
Display sqrt(-3).

I have a negative argument for square root

Display log(0).

I have a zero argument for log.

Display log(-6).

I have a negative argument for log.
```

EXAMPLE

To find the roots of a quadratic equation:

1. The quadratic equation is of the form

$$ax^2+bx+c=0$$

2. The program is as follows:

```
1.1 Demand a as "a".
1.2 Demand b as "b".
1.3 Demand c as "c".
1.4 Set d=b 2-4*a*c.
1.5 To step 2.1 if d=0.
1.6 To step 3.1 if d<0.

2.1 Set r(1)=(-b+sqrt(d))/(2*a).
2.2 Set r(2)=(-b-sqrt(d))/(2*a).
2.3 Display r(1), r(2) in form 1.

3.1 Set d=lsgrt(-d)/(2*a)|.
3.2 Set r(1)=-b/(2*a).
3.3 Display r(1), d, r(1), d in form 2.
```

Form 1:

```
Real Roots:R1= _ _ _ ._ _ _  ; R2=_ _ _ ._ _ _
```

Form 2:

```
Complex Roots:R1=_ _ _ ._ _ _ + _ _ _._ _ _J;
              R2_ _ _ ._ _ _ - _ _ _ ._ _ _ J.
```

NUMBER DISSECTION

Algebraic	Sign sgn(x)	*result=-1 if x 0*
		result 0 if x=0
		result +1 if x 0
Integer part	ip(x)	
fraction part	fp(x)	
exponent part	xp(x)	
mantissa part	dp(x)	

Note: x=ip(x)+fp(x)

x=dp(x)*10txp(x)

EXAMPLE

a) Set x=123.456.

Display ip(x), fp(x), x.

```
ip(x)=123        integer part
fp(x)=   .456    fractional part
  (x)=123.456
```

b) Set y=123.456*100.

Display y, dp(y), xp(y), ip(y), fp(y).

Results are:

```
       y=12345.6
dp(y)=      1.23456
xp(y)=      4
ip(y)=  12345
fp(y)=       .6
```

Note: y=1.23456*10↑4

decimal part or mantissa

exponent part or characteristic

c) Set a=123

Set b=-123.

Set c=0.

Display sgn(a), sgn(b), sgn(c).

```
sgn(a)=1     plus number
sgn(b)=-1    minus number
sgn(c)=0     zero
```

CONCATENATI0N

Concatenation, is the joining of two things by placing them next to each other. PeCos recognizes the symbol **&** for concatenation.

Textual expressions or variables may be concatenated.
The number of characters as the result of a concatenation must be 80 or less.

EXAMPLES

```
Set a="PeCos"
Set b=" Here "
Display a&b.
    a&b="PeCos Here"
Display a&b&""&a&b.
        "PeCos Here PeCos Here
```

Example

```
1.1 Set a="z".

1.2 Demand a(1) as "What is the first name".

1.25 Demand a(2) as "What is the last name".

1.30 Demand (3) as What is the street address".

1.35 Demand a(4) as "What is the city".

1.40 Demand a(5) as "What is the state".

1.45 Demand a(6) as "What is the zip code".

1.5 Do Part 2.

2.1 Erase.

2.2 Display a(1) & ", " & a(2) in form 1.

2.3 Display a(3), a(4)&" "a(5)&" "a(6) in form 2.
```

```
Form 1:

Name_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _


Form 2:
Address: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

LOGICAL FUNCTIONS

We learned that relational operators, are used to form logical expressions. The result of the evaluation of a logical expression is either true or false.

As a review:

```
2+2=5         is false
"cat" ="dog" is false
6>3           is true
```

Based upon the evaluation being true, we might do one thing and based upon the evaluation being false we might do something else. We have seen this in if clauses. What would happen if we wanted to look at several conditions as a group. As example, if the day of the week is, Monday, or Tuesday, or Wednesday or Thursday, or Friday we go to work.

Notice, the word **or,** is used between the various conditions.

Go to work if d=Monday, or d=Tuesday, or d=Wednesday, or d=Thursday or d=Friday, Where **d** is the day of the week. PeCos recognizes the word "or" as a logical function, and is interpreted as:

The or of two or more expressions is true if any of the expression is true.

an example is

a="Mon"

b="Tue"

y="Tue"

Display y=a or y=b.

y=a or y=b=true.

The first expression (y=a), is false, but the 2nd (y=b), is true. The or of these two expressions is true since 1 of them is true.

Now suppose we stated, "We play golf if it's Sunday, and the weather is sunny".

In this statement, the two conditions are related by the word **and.**

PeCos recognizes the word **and,** as a logical function, and is interpreted as:

The and of two or more expressions is true if, all the expressions are true.

an example is:

```
a="Sun"
b="Sunny"
y="Sun"
x="Sunny"
Display y=a and x=b.
        y=a and x=b=True.
```

Both conditions, are true, so the **and** of them is true.

Now set:

```
x="cloudy"
Display y=a and x=b.
y=a and x=b=false
```

since x=b is false

One more logical function, is the word "not". If an expression is true, the not of the expression is false, and if an expression is false, the not of the expression is true.

The 3 logical functions (or, and, not) can be used to make up complex logical expressions.

The order of evaluation, of a logical expressions with and or, not in it:

1. PeCos evaluates from left to right with
   a. Not first.
   b. Then or, and
2. Groupers of course can alter precedence.

EXAMPLE:

```
1.1 Display a or b, and not c or d.
1.2 Display ( a or b) and not (c or d).
Set b=false.
Set c=true.
Set d=6<3.
Do part 1  for a=true, false. a
or b and not c or d=true.
(a or b) and not(c or d)= false. a
or b and not c or d=false.
(a or b) and not c or d)=false.
```

If the logical functions are used in a mixed expression with numeric operators and relational operators, the rules of evaluation are:

1. From left to right with
   a. All numerical operators first (with the individual precedence rules obeyed).

   b. Then all relational operators.

   c. then not

   d. then and or

2. Groupers of course can effect precedence rules.

Example

```
1.1 Set a="Friday".
1.2 Set b=5.
1.3 Set d="z".
1.4 Demand d as "Day".
1.5 Demand t as "time".
1.6 To step 2.1 if d=a and t>b.
1.7 Display form 1.
2.1 Display form 2.
```

Form 1:

```
       The weekend has not yet arrived.
```

Form 2:

```
       It's time to go home for the weekend'.
```

TIMER

PeCos recognizes the word timer. Internally PeCos keeps track of time passed in seconds.

The value of timer is reset to 0 only when power is turned on (or the system is reset).

You can still use timer to keep track of time, it is useable as any other noun.

EXAMPLE

```
1.1 Set x=timer,

1.2 Set y=timer -x.        Sets y to number of seconds
                           passed since step 1.1 was
                           performed.

1.3 Display y, y/60, y/60/60.

1.4 To step 1.2 if y<3600.
```

Chapter 11

TAPE SYSTEM

Built into the main console of the unit are two tape recorder/play back units. These allow permanent storage of programs or data. (There is an optional package of two additional tape units available).

The unit on the right is designated as Tape number 0, and the one on the left side is Tape number 1.

There are 4 control keys for each individual tape deck From left to right the controls are:

REWIND-This allows you to rewind the tape. In this mode PeCos can not read or write to the tape.

F/F-Fast Forward, this allows you to advance the tape. In this mode, PeCos can not read or write to the tape.

ENGAGE-When a tape is placed in the lid, and the lid shut, this button will engage the head to the tape. It is in this mode only, that PeCos can read or write to a tape.

EJECT-If any other button is pressed, then on the first depression of this key all other keys will be released. On the second depression it will automatically lift the lid, so you may insert or remove a tape.

NOTE: Do not press any buttons, unless the tape lid is closed.

Also, located on the top rear of each tape deck is a mechanical counter, and reset button, (the same as on many standard audio tape decks).

As the tape advances, (or rewinds) this counter increases or decreases. This counter can be useful for keeping record of where on rapes, programs or data are stored. (The same way as the counter is used on audio tape systems). By knowing where something is stored, you can use the manual fast forward or rewind controls to bring the tape near the program you want.

EXAMPLE

You have written down the following locations of programs (you can even have stored this onto the beginning of the tape).

Program Name	Counter Location
Pay records	50
Check Book	175
Bills Payable	350

You would like to get to the check book program

1. Place the tape in the cassette unit, rewind the tape and reset the counter.
2. Use the fast forward, to counter location 175.
3. Now run the program to recall from the tape check book program. This will then be found very quickly.

A WORD ON TAPES

Only good quality cassette tapes should be used. Avoid purchasing discount tapes. Use C 60 type (30 minutes/side).

PeCos uses an optical reader to sense the difference between leader and tape passing the head. Do not use tape with opaque leaders, Try to use tapes with clear leader. Light pink will also work, but clear is the best.

TAPE MAINTENANCE

The only user maintenance required, is to occassionally clean the tape head. Oxides from the tape, or dirt can build up' on the tapehead, and interfere with reading or writing. Clean the head only with a standard Audio type head cleaning tape. Do not use any solvents or anything else on the heads.

NOTES ON USING TAPE SYSTEM:

1. When you are not using a tape, press the disengage button. Leaving a tape engaged for long periods of time can damage the tape.
2. Do not have more than 2 tapes running at one time, it's O.K. to have 1 tape running under PeCos control while you are rewinding a 2nd tape, but don't try to rewind more than 2 tapes at once.
3. Use one side of tape only.

TAPE OPERATING SYSTEM

The Main Features of the Tape Operating System are:

1. PeCos can start or stop any of the tape decks as long as the engage button, on the deck is depressed. It can only move the tape forward at 1^{7}/8 inches per second, (I.P.S.) rate. It is in this mode that PeCos reads or writes to a tape.
2. There are fast forward, and rewind buttons on each deck which you can use to advance or rewind the tape at about 40 I.P.S. In this mode PeCos can not read or write to that particular tape deck.
3. Each tape deck has a number (0 and 1 located in the main console). You can direct PeCos as to which tape unit it should read or write to by referencing these numbers.
4. PeCos deals with tapes by writing and reading on two tracks. One track is used for putting down addresses or block numbers. (See formatting tapes). The other track is where programs or data is written. By doing this PeCos always knows where it is on the tape. Each record can hold up to 80 bytes of information. A C 60 tape, can have about 1000 records, so a C 60 tape can hold up to 80000 bytes of information.
5. The PeCos system allows you to store or read from tape.

   a. 1 record worth or

   b. many records worth.

6. You can start to write or read any where on a formatted tape. You don't have to start at the beginning.

7. There is a tape in motion lamp on the front panel. This will be on whenever PeCos is reading or writing to a tape. Actually, the lamp will blink on and off since PeCOs would read the contents of a record, stop the tape store the information read, then start the tape again, and read the next record.

FORMATTING TAPES

1. All tapes used on PeCos, must be first formatted. Only tape number 0 is equipped to format tapes.

2. The statement to format is:

Format N records.

a. N is a number or numerical expression

b. This statement puts N records onto the tape in unit number O.

c. This is a direct command only.

d. The tape must be fully rewound, with the leader in front of the head. If not PeCos will respond with:

Please rewind the tape.

3. If you ask to format more records, than can fit on a tape, PeCos will format the entire tape, and when it sees the leader at the end, it will stop and say, End of medium.

EXAMPLE

Format 100 records.

This will write 100 address blocks onto the tape in unit number O.

4. Record-Record is a key word whose value, is the last record number written or read from: It can be used , like any other numerical noun.

EXAMPLE

Format 20 records.	*Tape in unit 0 starts, and tape in motion light blinks.*
Display record. record=19	*Note: record=19 since the first record is number O. 20 records consists of number 0-19.*

FILE:

1. The verb **file,** is for storing programs, parts, steps, forms.

2. The simple form of a file statement is:

   File noun.

   The *nouns* could be:

   a step number, part number, form number, all parts, all steps, all forms, all (meaning all parts, and forms).

Example

File step 1.1,

File part 6.

File all.

File part 6, step 2.1, form 12, form 13.

3. File can be either direct or indirect.

4. An option on the file statement is to specify a tape unit number.

Example

File all parts on tape 1. *Tape 1* ***will be selected***

Example

File part 1. *If no tape unit is specified PeCos will assume you want tape O.*

5. You can file something and give it a textual name.

EXAMPLE

File part 1 as "Payroll". ***Part 1 is filed on tape with the name "Payroll".***

Set a="Directory". *Name can be textual variable*

File part 16 as a.

File part 16 as a on tape 0.*Of course, you can still specify which tape to file on, and you ' may have an if clause.*

File part 16 as a on
tape 0 if y= YES".

RECALL

1. The verb recall is used to read in anything that was filed.

2. The simpliest form is:

   Recall.

   This will cause PeCos to read in from tape 0, the next file of information. (Whether it is a named file, or not).

3. When PeCos has finished recalling a file it will display. Done.

Example. (place formatted tape in unit 0.

```
Form 1.
    This is a sample.
File form 1.
turn power off/on.
Rewind the tape, and depress Engage
Recall.                         (Tape starts, and light blinks)
Done.
Display all.
Form 1.

    This is a sample.
```

5. You can optionally specify which tape unit to recall from.

   Example

   Recall,

   Recall from tape 1.

6. You can also have an optional If Clause.

7. You can recall a named file. In this case PeCos will look at the name stored in front of each file. If it finds a name that you requested, it will say found, then read the file, and when finished reading it says DONE.

If it finds a named file, which is not the one you requested, PeCos Displays:

File="Name of file found"

Still looking.

(And will continue to look for the file name you requested).

EXAMPLE

Reset PeCos

Place a formatted tape in unit 0.

1.1 Display 1.1.

1.2 Display 1.2.

2.1 Display 2.1.

```
File part 1   as "one".

File part 2 as "two".

The tape now has two named files on it.

Rewind the tape.

Reset PeCos                  (Clears all stored programs).

Recall "two "

File=One                     PeCos found a file named one which
                             is not what was requested
Still looking

Found                        PeCos found requested file, and
                             is finished reading it.
Done

Display all.

2.1 Display 2.1.


Reset PeCos

Rewind tape.

Recall.                       Ask to recall next file PeCos
                              recalls next file even though it
Done.                         is a named file and no request to
                              recall by that name
Display all.

1.1 Display 1.1.

1.2 Display 1.2.
```

It should be noted, that both the recall, and file statements can be used with the tapes anywhere. You don't have to be at the beginning.

WRITE, READ, LABEL FIND.

1. These statements are used for writing, and reading data only to a single record.
2. Write **a.**

   This will write the value of variable **a** to the next block on tape; a can be textual, or numeric, or logical and up to 80 characters.
3. Read **b.**

   Will read the next block from tape, and assign it to the letter **b**. This is exactly like a set statement except the value is read from tape.
4. A read or write statement can have optional **If clause**
5. You can specify which unit to read from or write to.

   Write **a** on tape O.

   Read b from tape 1.
6. Label as "name". The verb **label,** allows you to put a textual marker on the data tape.
7. **Find "name".**

   Will read the data track till it finds a textual marker the same as **name.**

Example

We will set up a file on tape with,names, and ages of various people.   First the program to input the names. and ages to PeCos.

```
Set b="b".
1.1 Demand a as "How many names are there".
1.2 Do part 2 for i=1(1)a.
2.1 Demand b(i) as "The name is".
2.2 Demand c(i) as "The age is".
```

Parts 1, and 2, will set array b with the names and array c, with the corresponding ages.

Now to save this on tape, we will write out 2 sections, the first with the names, and the second with the ages.

```
3.1 Label as "names".
3.15 Write a.
3.2 Do part 4 for i=1(1)a.
3.3 Label as "ages".
3.4 Do part 5 for i=1(1)a.
4.1 Write b(i).
5.1 Write c(i).
```

Part 3. First writes out a textual marker called "names" then it writes the value of **a** which is the number of names Then it has part 4. write out the names, in consecutive records following the label. Next part 3 writes out a textual marker called "ages". Then it has part 5 write out the corresponding ages.

Now, how do we get this information back into PeCos from the tape?

Rewind the tape and reset PeCos

```
6.1 Find "names".
6.2 Read a.
6.3 Do part 7 for i=1(1)a.
6.4 Find "ages",
6.5 Do ;part 8 i=1(1)a.
7.1 Read b(i).
8.1 Read c(i).
```

Part 6 first finds the label "names". We know from step 3.15 that the next record contains the number of names so we read this next record and assign the values to variable a. Next we read in the names to the array b. (the number of members of b will be the value of a). Next we find the label "ages" and read the following values into the array c.

Chapter 12

SAMPLE PROGRAMS

The purpose, of this chapter is to show how to write programs on PeCos.

Remember, the PeCos language is designed to help you with problem solutions, and not fight you. Many programs can be "lashed up", right at the console. Larger programs should be blocked out first. Define the main task, subtasks, formulas, etc. View each part of PeCos as being responsible for performing a subtask, use one part as the control or overall executive part. It might be a good idea to always use part 1, for this, so all your programs will start with a:

```
Do part 1.
```

Most programs, can be broken into four catagories, (and each catagory is broken into subtasks).

The catagories are:

1. Output headings, instructions, messages, and initialization
2. Getting the necessary inputs and data.
3. Performing the necessary calculations to obtain the solution
4. Finally outputting to the peripheral to get the solution

EXAMPLE

Future value of present amount, compounded monthly.

Problem:

For a deposit in a bank, we would like to find out how much we will have after a certain number of years.

The interest is compounded monthly.

Solution:

1). First we have to know the formula for future value.

it is:

$$F= P(1+ \frac{i}{12})^{12n}$$

where F=Future value

P=Present value

i=Yearly interest rate (in decimal form)

n=number of years

2). Next let's divide the program into the various subparts we need.

1. A control part to direct the "flow" of the program
2. A part to input P, i, and n.
3. A part to calculate F.
4. A part to display F.

3). Part 2-To input, P, i, and n.

2.1 Display ¦ .

2.2 Demand p as "Present value".

2.3 Demand i as "interest rate".

2.4 Demand n as "number of years".

Let's see how part 2 works.

Do part 2.

Present value=200

Interest rate=6

Number of years=3.5

Display P, i, n.

```
              P=200
              i=6
              n=3.5
```

Note that the interest rate is input as a whole number. Our formula wants a decimal so we will have to adjust i, after it is input.

4). Part 3.

To do the calculation

3.1 Set I=i/100. *(convert i to decimal)*

3.2 Set F=P*(i+I/12)?(12*n).

5). Part 4. To output to the display the result

Let's first define a form.

Form 1:

Future value $_ _ _ _ _ ._ _

We allow a field value with 7 digits, (2 to the right **to** the decimal point).

4.1 Display F in form 1.

6). Now the control part.

1.1 Do part 2.

1.2 Do part 3.

1.3 Do part 4.

Type in

```
Do part 1.
Present value=1000.
Interest rate=7
Number of years =2
Future value $1149.81
```

Suppose now we would like to make some changes to the program, Very easy since we have written our program in parts.

Let's change the output part, so it gives a better description of what the problem has done.

```
Form 2:
For a present value of $_ _ _ _ _ _._ _
At an annual interest rate of _ _ ._ _ %
Form 3:
Compounded 12 times a year
For _ _ years.
```

Now display part 4.

```
4.1 Display F in form 1.
```

and add the following

```
4.01 Erase.
4.02 Display P, i in form 2.
4.03 Display n, in form 3.
4.04 Display _ .
```

*Now try the program. Try **some** other changes, by adding **steps,** or deleting steps, or rewritting steps.*

Example 2

Saving the program of example 1, on tape.

Problem-We now want to file the program on tape so if we shut off power, we will still have the program available by simply recalling it from tape.

A. Solution

1. Place a formatted tape into unit number 0.
   Rewind the tape, and reset the counter. Since this is not a very long program, and we will probably put more than one program on the tape.
   Let's plan to write a directory of what is on the tape.

2. Advance the tape, (using the fast forward), till counter reached 30. Up to 30 we will later write the directory.

3. Now, what do we want to file on the tape? We want to file all parts, and all forms. (We can use the word all, which means all parts, and all forms). We also, want to name the program.
   Let's call it compound interest.

4. Press the engage button, of tape 0, and type in:
   File all as "compound interest".
   The tape starts, and PeCos is writing the file name, (compound interest), all parts, and all forms.

5. When the tapes stop, note the counter number. Now rewind the tape, and let's write out the directory.
   First, we will put a label out, what the data we are writing is.
   Label as "Directory".
   (We can later look to find this label)

Next, let's write out the number of programs we have on tape (1).

Write 1.

Now we will write out the program name, its starting counter number, and ending counter number.

Write "Compound interest"

Write 30.

Write 45.

We are now finished saving the program as well as all information about where it is stored.

B.

1. Shut off power and then turn it on.

Display all.

There is nothing in PeCos, so let's recall our program.

2. Place the tape in recorder # 0, and rewind it.

   Reset the counter.

   Since we only have one program stored on the tape, we could have written the name, and starting counter number on the tape label, in this case we don't need the directory we wrote in the front.

   Advance the tape to counter # 28. (a little before where our program is written).

   Now type in:

   Recall "compound interest".

Make sure, you type in the name exactly as we did in the file statement. If there is one letter different, (ie, lower case when we used upper case etc.), or an extra space, PeCos will not recognize it as the name stored on tape. If you typed in the name correctly, PeCos will respond with:

Found.

This means, it found the named file, and is now reading it in. When PeCos is finished recalling it will display

Done.

Now type in:

Display all.

You can see that our compound interest program is now back in PeCos, (and still on tape).

3. Suppose, we didn't know what was stored on the tape. We can use the directory. Clear PeCos by turning power off/on.

The program to read and display directory is:

```
100.1  Find "Directory".
100.15 Display ¦ , form 100.
100.2 Read a.
100.25 Display a in form 101.
100.3 Set x=1.
100.35 Read n.
100.4 Read f.
100.45 Read e.
100.5 Display n, f, e in form 102.
100.65 Done if x=a.
100.7 Set x=x+1.
100.75 To step 100.35.
```

EXAMPLES-Continued

Form 100:

Directory

Form 101

There are _ _ programs on this tape.

Form 102

Program name:_ _ _ _____ _ _ _ _ _ _ _ _ _

Starts at # _ _ _   Ends at # _ _ _

Step 100.1   Finds the label "Directory", that we wrote out. This is to make sure we have a tape in the unit, which has a directory on it.

Step 100.15 Displays a heading.

100.2 Reads the next block after "Directory" and assigns the value read to the variable named a. We know that this value is the number of programs on the tape. (we know this because, that is how we wrote the directory).

Step 100.25 Displays the number of programs in form 101

Step 100.3 We set x to a value of 1 x will keep count of how many program names we read in from the tape.

Steps 1.0035, and 100.4 and 100.45 read 3 consecutive blocks from the tape, and assign the values read to the variables n, f, and e. We know the values will be the programs name, the starting counter number and ending counter number.

Step 100.5 Displays the values of n, f, and e in form 102.

Steps 100.65, and 100.7, and 100.75 check to see if we have read the entire directory. If we have, we are done. If not then we go read 3 new values for n, F, and e, and display them.

Now type:

Do part 100.

The display will show

```
                   Directory.
There are 1 programs      on this tape.
Program name:   Compound interest.
Starts at # 30            Ends at # 45.
```

4. If we want to add more programs to the tape. We first will advance the tape to 50 (leave some room between the end of compound interest and the start of the next program).
   Then file the program, and note, the counter at the end.
   To up date the directory, we have to change the value after the label "Directory". (Which is the number of programs on the tape). Then pass over any already written names, (count records passed), and write the next program name with it's corresponding counter number.

EXAMPLE 3-Learning about PeCos

Problem:

You've read the entire book, and there are some things that are not clear. You're not sure how PeCos will react and what it will do if you type in certain statements. Should you call or write APF?

Solution:

No! Experiment first. The PECos language is designed so you can easily experiment, and see the results. With it's error diagnostics, PeCos will always try to help you. You can't hurt PeCos by typing in incorrect statements. If you get it lost in some loop, you can always press the interrupt key. You can easily insert and delete steps. (Such as Stop).

As an example:

You are not sure how loops work

Type in the following:

```
1.1 Set x=1.
1.2 Display form x.
1.3 Done if x=4.
1.4 Set x=x+1.
1.5 To step 1.2.
Form 1;
Hi! This is form 1.
Form 2:
Hi! This is form 2.
Form 3:
Hi! This is form 3.
```

Step 1.5 then sends PeCos back to step 1.2. where it displays form x, again, but now x is=2., so PeCos displays Form 2.

PeCos will continue this loop of step 1.2-1.3-1.4-1.5-1.2 etc. until x=4 and then it is done.

Example

Let's add a step to part 1.

```
1.25 Stop,
Now Do part 1.
                   Hi! This is form 1.
Stopped by step 1.25
```

Now type in:

```
Display x.
        x=1
```

PeCos did step 1.2 with x=1

Next Type:

```
Go.
                   Hi! This is form 2.
Stopped by step 1.25
```

Notice that the Go statement, returned PeCos to the step after where the stop statement occurred. Go always has PeCos return to a program that was halted by a Stop, interrupt or error.

Re enter step 1.5

```
1.5 To step 1.01.
```

We have entered 1.5 incorrectly, (there is no step 1.01).

Now type in Go.

```
Form 4:
Hi! This is form 4.
Do part 1.
Hi! This is form 1.
Hi! This is form 2.
Hi! This is form 3.
Hi! This is form 4.
```

Step 1.1 Sets x=1.

Step 1.2 Says display form x.

PeCos see this and looks for the value of x. Right now it finds x=1 (from step 1.1), so it substitutes 1 for x. This means Step 1.2 says display form 1. (which it does).

Step 1.3 says we are done if x=4. (which it is not). So we go on to step 1.4 which says set x=x+1. Remember in a Set statement PeCos looks to the right of the equal sign and does an evaluation of the expression. It finds x is 1 so x+1 is 2. It then sets the variable on the left of the=sign to this value.

(So now x becomes 2)

Try:

```
Display x
        x=4.                  (last value of x).
```

Now try.

```
Do step 1.4.
Display x.
        x=5.
```

Try it again, and again, See that x keeps going up by one.

```
I'm at step 1.5.
I can't find step 1.01.
PeCos has found an error in step 1.5.
Display step 1.5.
1.5 To step 1.01.
```

Type in

```
1.5 To step 1.1.
```

Type Go.

```
                  Hi! This is form 3.
Stopped by step 1.25.
```

Notice that this time when GO was typed in PeCos went back to the step that had the error. Also notice that when the error message occurred, we were able to do the direct command

Display step 1.5.

We could have done any direct commands, and then typed Go.

Now try:

Example 4. Alphabetizing and Sorting

Problem:

We have a list of words which we would like to rearrange in alphabetical order.

Solution:

1. First to divide the program into subparts.

a. Part 2, to input the list of words.

b. Part 3,to rearrange the words into alphabetical order.

c. Part 4, to display the list of words in alphabetical order.

d. Part 1, as the control part and miscellaneous functions.

2. How many words will be input? Why don't we ask that to be input first.

2.1 Demand N as "How many words are there" .

Now, we simply need a step that demands the next word. We want to do this step N times

10.1 Demand w(J) as "The word is".

2.2 Do part 10 for J=1(1)N.

Step 2.2 says to do part 10 for J=1, then it does it again for J=2, etc. until J=10. As we type in the words from the keyboard, each word becomes the value of a member of the array w, (w(1), w(2) etc.).

We can try part 2, but first we better define that array will get textual values.

```
2.15 Set w="z
Now try Do part 2.
How any words are there?=4
The word is=Cat
The word is=Dog
The word is=House
The word is=Car
Now,
Display w.
    w(1)=Cat
    w(2)=Dog
    w(3)=House
    w(4)=Car
```

3. We now have the array w with the words in the order they were input. To sort them alphabetically, we do what is called a bubble sort.

The routine will be:

```
3.1 Set I=0.
3.15 Set I=I+1.
3.2 Set J=0.
3.25 Set J=J+1.
3.3 To step 3.5   if w(J) = w(J+1).
3.35 Set z=w(J).
```

```
3.4 Set w(J)=w(J+1).
3.45 Set w(J+1)=z.
3.5 To step 3.25 if J = N-I.
3.55 To step 3.15 If I = N.
```

This routine causes the array w to be rearranged, so that the lower values (in dealing with text ***a****, has a lower value than* ***b*** *etc.). bubble to the top .*

Step 3.3 compares 2 members of the array, steps 3.35, 3.4 and 3.45 swaps the members if upper members (w(J) has a higher value than the lower member (w(J+1).

```
4. Part 4, will display the alphabetical list.
   4.1 Display w(I) in form 1.
Form 1:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

```
5. Now for the controlling part.
1.1 Erase.
1.2 Do part 2.
1.3 Do part 3.
1.4 Do part 4  for I=1(1)N.
```

Now let's try the program

```
How many words are there=5.
The word is zebra
The word is dog
The word is house
The word is cat
The word is apple

            apple
            cat
            dog
            house
      zebra
```

EXAMPLE 5.

Problem: Given a year/month and date, what is the day of the week?

Solution:

1.   First the headings and instructions.

2.1 Display ¦ , form 1,_,form 3, form 4.

Form 1:

```
                     Day of week
```

Form 2:

Enter the year, month number (ie, Jan=1, Feb=2, etc) and Date

Form 3:

And I'll tell you the day of week.

2. Next to get the inputs.

3.1 Demand y as "year".

3.2 Demand m as"month number".

3.3 Demand d as "date".

3. Now for the calculation.

We compute the "day number" (0-6) with the following formula.

4.3 N=d+2*mfip(.6*(M+1))+y+ip(y/4)-ip(y/100)+(ip(y/400)+2.

4.4 N=ip((N/7-ip(N/7))*7+.5)

If the month is January or February, there is a correction factor required.

4.1 To step 4.3 if m 2.

4.15 Set m=m+12.

4.2 Set y=y-1.

4. Now that we have the day number, we have to display the day.

First to set up an array with values of "Sat", "Sun," etc.

5.1 Set c(0)="Sat”.

5.2 Set c(1)="Sun".

5.3 Set c(2)="Mon".

5.4 Set c(3)="Tues".

5.5 Set c(4)="Wed".

5.6 Set c(5)="Thurs".

5.7 Set c(6)="Fri".

We will only have to do this part, the first time we run the program. After that the value of members of c, are set.

To display the solution.

```
6.1 Display c(N) in form 4.
```

Form 4:

```
The day is _ _ _ .
```

Step 6.1 uses the value of the day number (N) as the subscript of the array c. This will give us the correct value to display.

5. Now for the control part.

```
1.1 Do part 5.                (Set up values for array c).
1.2 Do part 2.
1.3 Do part 3.
1.4 Do part 4.
1.5 Do part 6.

    Do part 1.
```

```
                           Day of Week
Enter the year, month number, (ie Jan=1 Feb=2, etc) and
date.
And I'll tell you the day of the week.
Year=1953
Month=3
Date=4
The day is=Wed.
```

7. Now suppose we would like to check for inputs of non-existent years, months, or dates.

LEt's modify part 3.

```
Display part 3.
3.1 Demand y as "year".
3.2 Demand m as "month number",
3.3 Demand d as "date".
Form 5:
That is an illegal entry-try again.
3.11 Display form 5 if fp(y)?0 or sgn (y)?1.
3.12 To step 3.1 if fp(y)?0 or sgn(y)?1.
```

Steps 3.11 will display form 5 and send the program back to step 3.1 if y is not a whole number (ie the fractional part is not=0), or if y is not positive (sgn(y)?1)

```
3.21 Display form 5, if fp(m)?0 or sgn (m)? or m>12.
3.22 To step 3.2 if fp(m)? 0 or sgn (m)? or m>12.
```

Steps 3.21 and 3.22 not only check the sign and fraction part of m, but also check that m, is not greater than 12.

To check the date is a little bit more difficult. (It depends on the month and if it's a leap year). We'll do a sub-part.

```
7.1 * to check if legal date
7.15 Set z=31.
7.2 Set z=30 if m=4 or m=6 or m=9 or m=11.
```

7.3 Set z=28 if m=2 and fp(y/4)?0.

7.4 Set z=29 if m=2 and fp(y/4)=0.

7.5 Set z=0 if fp(d)?0 or sgn(d)?1 or d>z.

7.6 Display form 5 if z=0.

3.31 Do part 7.

3.32 To step 3.3 if z=0.

Steps, 7.15, 7.2, 7.3, and 7.4, will set z equal to the number of days in the month,

Step 7.15 assumes it is a 31 day month, and so sets z=31.

Step 7.2 will set z equal to 30, if the month number is 4, 6, 9 or 11, (april, June, September, or November).

Step 7.3 sets z to 28 if the month is 2(Feb) and the year number is not divisible by 4.

Step 7.4 sets z, equal to 29 if the month is February and the year is divisible by 4, (a leap year).

Next step 7.5 sets z=0, to if the date is illegal. If it is a legal date it leaves z with what it was previously set, (ie, 31, 30, 29, or 28).

Step 7.6 Displays form 5. If z=0 indicating an illegal date. When the program returns to part 3, (step 3.32), the value of z tells it whether there was an illegal date or not.

APPENDIX A.

POWER UP CONDITIONS

Whenever, PeCos is powered-up, or (Reset), the following conditions will occur:

1. The monitor will be cleared except for the top line, which will read:
   PeCos Here
2. The power on lamp will be lit.
3. The keyboard ready lamp will be lit.
4. The input request, and tape lamps will be out.
5. Internally, memory is cleared. There will be no programs stored, variables, or functions defined.

APPENDIX B

Glossary of PeCos Language

Step number	Command	Nouns	Modifiers
1.23	*Display*	*x, y, z+3*	*in form 3 if x+y 10.*
1.4	*Do*	*part 6*	*for x=1(10)100, 1000*

Term	Meaning	Example
Direct Statement	Step number not present: Command is executed immediately.	*Display 1+2.*
Indirect Statement	Step number is present: Command is stored in order of step number.	*1.12 Display x,y,z*
Step	A stored command:step number is limited to 9 digit number >1.	*1.1234-step 1.1234.*
Part	A group of steps whose step numbers have the same integer parts.	*1.123* *1.222 part 1.*
Form	Pictorial representation of literal information, specifying how an output should appear, may have fields which will be filled in with values.	*Form 1:* *Payroll*
Field	A string of underscores, (with an optional decimal point), or a string of periods. Used to designate regions of a form to be filled with values.	*Form 8:* Gross pay$___.__ Net pay$____.__
Numbers	9 significant digits: A number greater than $10^{-99}$and less than 9.99999999*10^{99}.	*0, 123, 3.4*10199.* -6.1*10?41,*3.1*10?(-9)*
Symbols	Single letter identifiers-upper, or lower case. May identify a numeric value, (numbers), logical values (true, false) textual values, (letter, numbers, and symbols), arrays of values or formulas.	*a, b, c, ...x, y, z* *A, B, C, ...X, Y, Z*

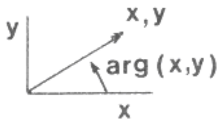
Array	An arrangement of 1 or more values grouped together by using a common symbol **as** their name.	_Array i_ _i(1)=a+b+6._ _i(2)=log(46)._
Arithmetic	Exponentiation (?), multiplication. (*), Division (/), addition (+), and subtraction(-give true results rounded to 9 digits.	_2?6+3-9*8/3_
Relations	<.>,<,>, =, ?, used for comparison to give a true,false result.	_6<9 (true)_ _2+3=5 (true)_ _3<4<2 (false)_
Logic	And, or, not used to compare logical expressions.	_2>3 or 4<3 (false)_ _2<3 and 2<4 (true)_
Text	Any group of letters, numbers, symbols enclosed in a pair of quotes.	_"My name is PeCos"_ _"How much is 2+2?"_
Groupers	( ), or [ ], used interchangeably in pairs.	_(3+1/2)+[1/4]*5_
Expressions	A representation by symbols of a value or relation. An expression may be numerical, logical or textual.	_Numerical: 2+3*5-6/8_ _Logical: 3<4>1_ _Textual: "My name is PeCos"._
Conditional Expressions	[A:B;C]A is a logical expression, if A is true, use the value after the colon, if A is false use the value after the semicolon.	_[1+2=4."Right";"Wrong_ _=Wrong._

Term	Meaning	Example
Cancel	Terminates execution of all Do's Can have an if clause.	*Cancel if size < 50.*
Delete	Clears from memory parts, steps, forms, values, Can have an if clause.	*Delete x, part 3, all forms.* *Delete all values if a="done".*
Demand	Displays an identification and and equals sign, then waits, for the user to input a value. Can have an if clause.	*Demand a as "a".* *Demand r as "rate".*
Display	Display on CRT, values, parts steps, text, blank lines. Can have an if clause.	*Display x+3, D(1), all steps.* *Display "The quick brown fox".* *Display a, b, in form 6 if b<c.*
Do	Initiates exection of step or part, (step by step beginning at first step of part). Can repeatedly execute step or part if modified by a for clause or times clause. Can have an if clause.	*Do part 6.* *Do step 10.21,* *Do part 3 for i=1(2)10, 20.* *Do part 20 for i=true*
Done	Terminates execution of current Do for current repetition. Can have an if clause.	*1.1 Set a=sqrt (a).* *1.2 Done if a 1.* *1.3 To step 1.1.*
Erase	Clears the screen. Can have an if clause.	*Erase if N=16.*
File	Used to store steps, parts, forms on tape. Can optionally specify which tape to use. Can give a name to the file, and can have an if clause.	*File part 1.* *File all forms on tape 1.* *File all parts, as "Payroll".* *File all parts as "Payroll" on tape 1 if a=6.*

Term	Meaning	Example
Find	Will read tape, and look for a textual name. Can specify which tape to **use,** and have an if clause.	*Find "Pay Records".* *Find "names" from tape 1 if p.*
For	Modifies a Do command only. PeCos executes step or part for specified set of values	*Do part 1 for i=1(10)100.* *Do step 6.1 for J=1,2,* *6, 7, true.*
Form	Gives a description of how an output should appear. May **have** fields or not. A form can be displayed, or used to modify a display statement.	*Form 3: (Touch E/E)* *Payroll (Touch E/E)*
Go	Continues execution after interrupt error message or stop command. Direct command only.	*1.1 Set b=3*g.* *1.2 Display b.* *Do part 1.* *I'm at step 1.1* *g=???* *g= 4* Go.
If	Modifies any command. PeCos carries out command if condition is true.	*Display x, if u<x.* *Set y=3 if x<10 and x*g=10.*
In form	Modifies a display command only. PeCos displays values in fields specified in form.	*Display a, b, c, in form 3.*
Label as	Will place a textual name on to tape. Can specify which tape and have an if clause.	*Label as "deductions"*
Print	Outputs to the printer, values parts, steps, text, blank lines and paging, the statement may contain an if clause and/or in form clause.	*Print x+3, 0(1) all steps.* *Print "The quick brown fox".* *Print a, b, in form 6 if b<c.*

Term	Meaning	Example
		1.1 Display i. _1.2 Quit if i=50._
Read	Will read the next value from tape (up to 80 characters), and assign it to a symbol. Can specify which tape to use, and can have an if clause.	_Read a._ _Read b from tape 1._ _Read c from tape 0 if p="ready"._
Recall	Retrieves stored file from tape. Can recall a named file. Can specify which tape to recall from. Can have an if clause.	_Recall._ _Recall "Payroll"_ _Recall from tape O._ _Recall "Payroll" from tape 1 if a="Ready"._
Set	Assigns a value to a single lower or upper case letter.	_Set a=3._ _Set b="John Smith"_ _Set c=a<5._ _Set d=14 if a > 5._
Size	Number of storage units currently available. 1864 are available upon power initialization or reset.	_Display size._ _Do part 6 if size 500_
Stop	Suspends step by step execution and returns control to keyboard. Can have an if clause.	_10.1 Demand a._ _10.2 Stop if a=null._ _10.3 Set p=2*a._
Timer	The number of seconds since power initialization or reset.	_Display timer._ _Set a=timer._ _Display timer-a._
To	Alters step by step sequence. Continues at indicated part or step. Can have an if clause.	_To step 3.5._ _To step 1.1 if p=true._
Write	Will write a sinlge value up to 80 characters in the next record on tape. Can specify which tape to use and have an if clause.	_Write 1+2._ _Write "John Smith", on tape 1._ _Write a b on tape 1 if p=”end”._

INTERNAL FUNCTIONS

Symbol	Meaning	Example
sqrt(x)	*square root: x>0.*	sgrt(9)=3.
sin(x)	*trig sine:* |x|in *radians 100.*	sin(?/6)=.5
cos(x)	*trigcosine:* |x|in *radians 100.*	cos( )=1.
log(x)	*natural log x: x>0.*	log(2)=.69314718
exp(x)	e^x	exp(1)=2.71828182
arg(x)		
sgn(x)	*alegebraic sign: -1 for x<0,* *0 for x=0; 1 for x>0·.*	sgn(-123)=-1.
ip(x)	*integer part*	ip(12.34)=12.
fp(x)	*fractional part*	fp(12.34)=.34
dp(x)	mantissa *part*	dp=(12.34)=1.234
xp(x)	*exponent part*	xp(12.34)+1
|x|	*absolute value*	|-6| =6
&	*concatenation*	"PeCos" & “Here”= “PeCos Here”

APPENDIX C

Summary of Error Comments

When you do something that PeCos does not understand, or can not do, PeCos tells you about it with an ***Error Comment.***

If an error occurs in a direct statement, you must correct the error, and retype the statement. If any error occurs in an indirect statement, you can correct the error and then type Go. PeCos will continue.

The comments are usually self explanatory, but below is a check list of what the cause of the error was and how to correct it.

A). Eh?

This comment, occurs when PeCos does not understand a statement, and has no idea of what you are trying to state. It usually occurs when you have typed in a statement which violates basic rules of sentence structure.

To correct the error, you have to retype the statement. A check list of possible causes is:

1. Initial capital missing.
2. No final period.
3. Spaces missing where needed.
4. Spaces added where not allowed, (within a number).
5. Misspelling of a word.

A check list of possible causes-continued

6. Unpaired parenthesis, or brackets

7. "El" instead of 1 or vice versa.

8. "Oh", instead of *0* or vice versa

B). Errors dealing with numbers

The following errors occur if numeric value is used which out of PeCos' range.

The only solution is to correct the value to a legitimate range.

Please limit numbers to 9 significant digits.

I have an overflow.

I have a zero divisor.

I have a negative base to a fractional power.

c). Errors with functions:

I have an argument=0 for log.

I have a negative argument for sqrt.

Please keep |x| < 100 for sin(x) or cos(x).

D). Errors with steps, parts, forms

Please limit step labels to 9 significant digits.

Please limit numbers to 9 significant digits.

Form number must be an integer between 1 and 10?9.

I have too many values for a form.

D). Errors with steps, parts, forms-continued

I can't express values in your form. I need individual values for a form. I can't find part #.

I can't find form #.

I can't find step #.

E). Errors with variables or arrays.

Letter=??? *means a variable or array name used for which there is no definition.*

Subscripts must be an integer between -999 and +999.

Please limit number of subscripts to 10.

I can't assign different data types to the same array.

I can't change the number of subscripts of an array.

F). Errors with iterations.

I can't find step # for iteration.

Illegal set of values for iteration.

G). Tape

1. I couldn't read the record number.

I couldn't read the information.

G). Tape-continued

If an error occurs in reading a file, rewind the tape, prior to the error, and type in **Recall.** (no name). PeCos will continue, and try again.

2. No response from device.

This message occurs during 2 cases:

a. There is no tape in a deck selected, or there is an unformatted tape in a deck.

b. PeCos read an address number and found no data written in the record.

Please rewind the tape

This message occurs during formatting. If tape was not fully rewound.

I'm past the last record

The tape does not have any more records formatted.

End of medium

Occurs during formatting a tape if PeCos sees leader at the end of a tape.

H). Miscellaneous

Don't give this command directly

Occurs with a **to, demand, done** or stop

Don't give this command indirectly

Occurs with a **go, format, or cancel**

Stopped by step _ _ _

Indicates a stop statement was reached

Revoked by interrupt

The interrupt key was depressed. The command that was interruped is revoked and must be retyped to do again.

Interrupted at step _ _ _

The interrupt key was depressed while PeCos was doing a command The command can be resumed by typing in **Go.**

Please limit strings to 80 characters

Occurs if concatenation of strings results in more than 80 characters.

APPENDIX D

Useful Formulas and Equations

BUSINESS AND FINANCE

Future Value of an Investment

$$T=P(1+i/N)^{n*y}$$

T=Future Value

P=Initial investment

i=Nominal interest rate

n=number times compound/year

y=number of years

Annuity

$$T=R*\left(\frac{(1+i/n)^{n*y}-1}{i/n}\right)$$

T=Future Value

R=Amount of Regular deposit

N=number of deposits/year

y=Number of years

i=nominal interest rate

Nominal Interest Rate

$$i=N(T/P)^{1/n*y}-n$$

i=Nominal Interest Rate

P=Initial investment

T=Future Value

N=Number compounding periods/year

y=Number of years

Loans

$$R=\frac{i*P/n}{1-\left(\frac{i}{n}+1\right)^{-n*y}}$$

R=Payment

i=Annual interest rate

p=Principal

n=Number of payments/year

y=Number of years

BUSINESS AND FINANCE Continued

$I=B.\frac{i}{N}$

I=Amt of a payment towards interest

B=Balance

i=Interest rate (yearly)

N=Number of payments/year

A=R-I

A=Amount amortirized with each payment, (amount of each pay= ment toward principal)

L=R+(p-R*N*Y)

L=Amount of last payment

R=Regular payments

P=Principal

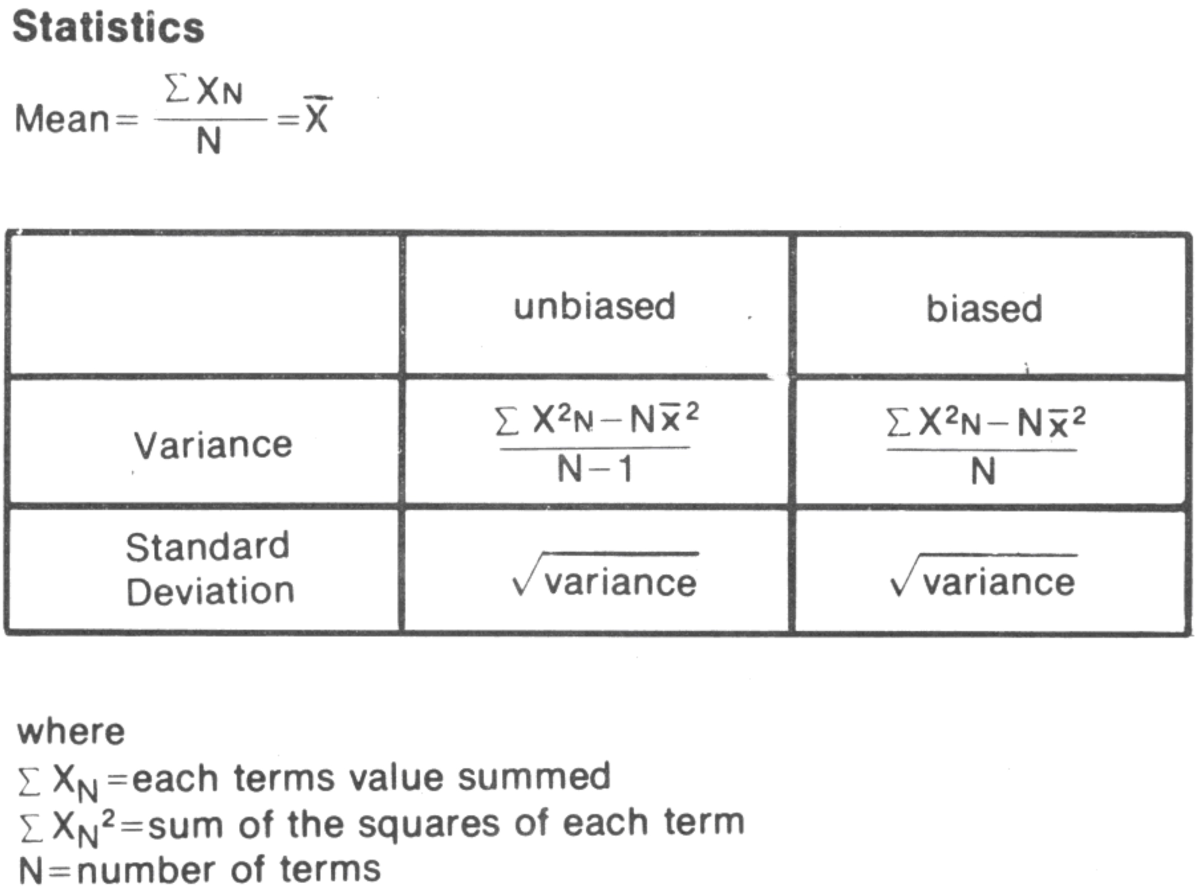
N=Number of payments per year

y=number of years

Interest Compounded

$A= P(1 + i)^N$

A=future value
P=present value
i=interest per period in decimal
N=number of periods
PMT=payments per period



Statistics

$$\text{Mean}= \frac{\sum X_N}{N} = \bar{X}$$

	unbiased	biased
Variance	$\frac{\sum X^2_N - N\bar{x}^2}{N-1}$	$\frac{\sum X^2_N - N\bar{x}^2}{N}$
Standard Deviation	$\sqrt{\text{variance}}$	$\sqrt{\text{variance}}$

where
$\sum X_N$=each terms value summed
$\sum X_N{}^2$=sum of the squares of each term
N=number of terms

Logarithms

$\log_a x = \frac{\log_{10} x}{\log_{10} a}$ converting log (any base) of x to $\log_{10} x$

$\log xy = \log x + \log y$

$\log (x/y) = \log x - \log y$

$\log (x^y) = Y \log x$

$\log_a b = 1/\log_b a$

Complex numbers

$x + iy = r(\cos\Theta + i \sin\Theta)$

where $r = \sqrt{x^2 + y^2}$

$\Theta = \mathrm{TAN}^{-1} \frac{Y}{X}$

Euler Identities

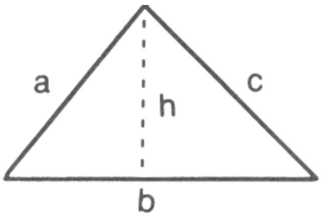
$e^{i\theta} = \cos\Theta + i \sin\Theta$

$e^{-i\theta} = \cos\Theta - i \sin\Theta$

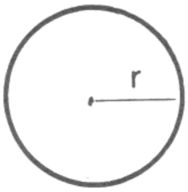
$i = \sqrt{-1}$

Geometry

Triangle: perimeter $= a+b+c$
area $= \frac{1}{2}(bh)$

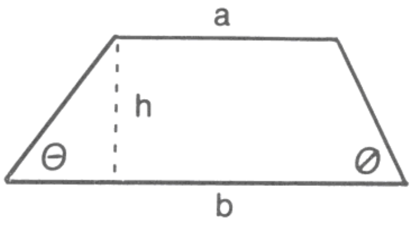


Circle: circumference $= 2\pi r$
area $= \pi r^2$



Analytic equations:

$$\frac{x^2}{r^2}+\frac{y^2}{r^2}=1$$



Trapezoid:

$$\text{area}=\frac{1}{2}h(a+b)$$

$$\text{perimeter}=a+b+h\left(\frac{1}{\text{Sin}\Theta}\quad\frac{1}{\text{Sin}\varnothing}\right)$$

Ellipse

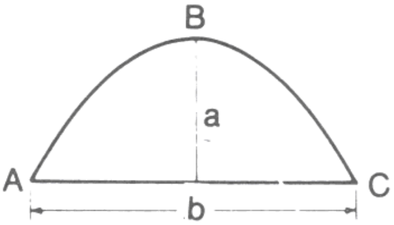
area $= \pi ab$

a = Semi-major axis
b = Semi-minor axis

Analytic Equation

$$\frac{X^2}{a^2}+\frac{Y^2}{b^2}=1$$

Parabola



$$\text{Area}=\frac{2}{3}ab$$

Arc Length ABC

$$=\frac{\sqrt{b^2+16a^2}}{2}+\frac{b^2}{8a}\ln\left(\frac{4a\sqrt{b^2+16a^2}}{b}\right)$$

Analytic Equation

$x^2 = \pm 2PY$

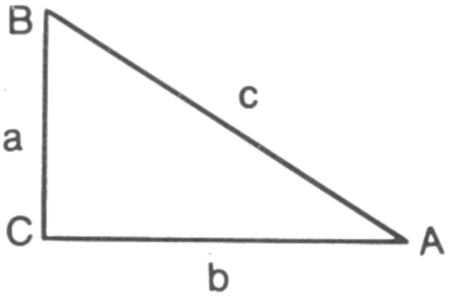
Sphere: area $= 4\pi r^2$ volume $= \frac{4\pi r^3}{3}$

Regular Polygon of N Sides

$$\text{Area}=\frac{1}{4}Nb^2\text{Cot}\left(\frac{\pi}{N}\right)$$

Perimeter $= Nb$.

Trigonometry



sin A	=	a/c	cscA	=	1/sin A
cos A	=	b/c	sec A	=	1/cos B
tan A	=	a/b	cot A	=	1/tan B

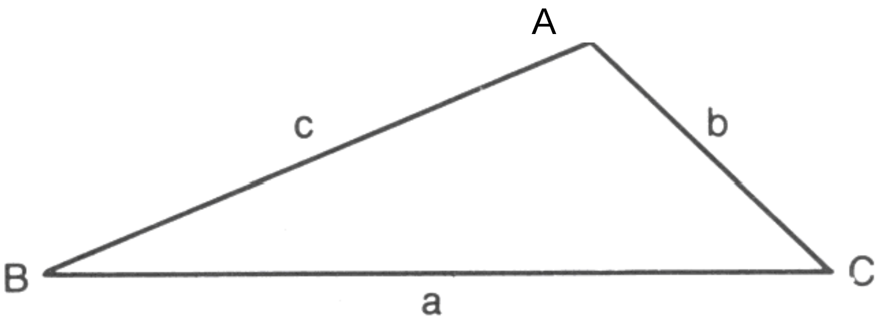
Relationships Among Trigonometric Functions

$$\sin^2 A + \cos^2 A = 1$$

$$\tan^2 A + 1 = \sec^2 A$$

$$1 + \cot^2 A = \csc^2 A$$

Relationships Between Sides and Angles of a Plane Triangle



law of sines:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

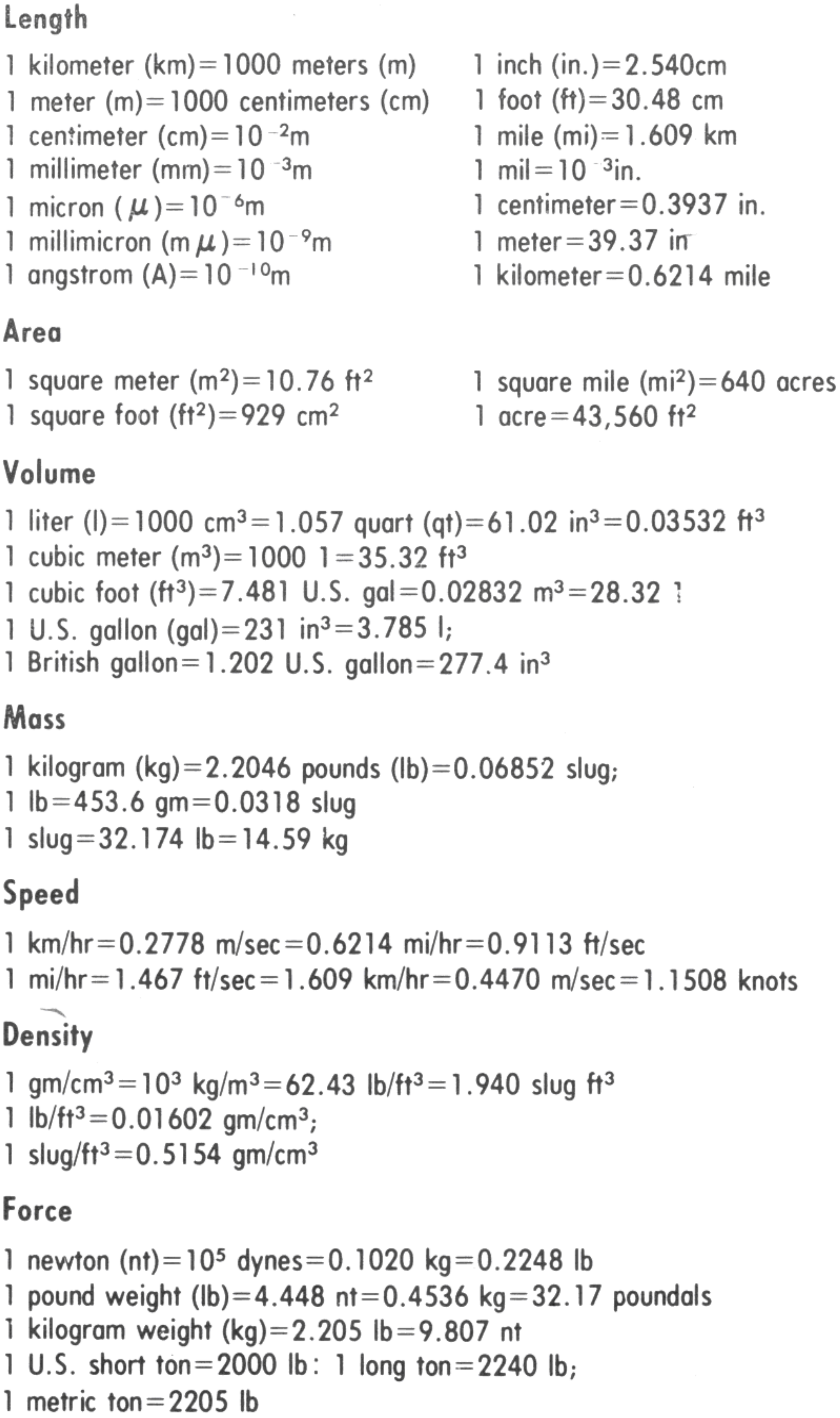
law of cosines:

$$a^2 + b^2 - 2ab \cos C = c^2$$

Law of Tangents.

$$\frac{a+b}{a-b} = \frac{\text{Tan } ½(A+B)}{\text{Tan } ½(A-B)}$$

Conversions Factors



Length

1 kilometer (km) = 1000 meters (m)

1 meter (m) = 1000 centimeters (cm)

1 centimeter (cm) = 10^{-2}m

1 millimeter (mm) = 10^{-3}m

1 micron (μ) = 10^{-6}m

1 millimicron (mμ) = 10^{-9}m

1 angstrom (A) = 10^{-10}m

1 inch (in.) = 2.540cm

1 foot (ft) = 30.48 cm

1 mile (mi) = 1.609 km

1 mil = 10^{-3}in.

1 centimeter = 0.3937 in.

1 meter = 39.37 in

1 kilometer = 0.6214 mile

Area

1 square meter (m^2) = 10.76 ft^2

1 square foot (ft^2) = 929 cm^2

1 square mile (mi^2) = 640 acres

1 acre = 43,560 ft^2

Volume

1 liter (l) = 1000 cm^3 = 1.057 quart (qt) = 61.02 in^3 = 0.03532 ft^3

1 cubic meter (m^3) = 1000 1 = 35.32 ft^3

1 cubic foot (ft^3) = 7.481 U.S. gal = 0.02832 m^3 = 28.32 l

1 U.S. gallon (gal) = 231 in^3 = 3.785 l;

1 British gallon = 1.202 U.S. gallon = 277.4 in^3

Mass

1 kilogram (kg) = 2.2046 pounds (lb) = 0.06852 slug;

1 lb = 453.6 gm = 0.0318 slug

1 slug = 32.174 lb = 14.59 kg

Speed

1 km/hr = 0.2778 m/sec = 0.6214 mi/hr = 0.9113 ft/sec

1 mi/hr = 1.467 ft/sec = 1.609 km/hr = 0.4470 m/sec = 1.1508 knots

Density

1 gm/cm^3 = 10^3 kg/m^3 = 62.43 lb/ft^3 = 1.940 slug ft^3

1 lb/ft^3 = 0.01602 gm/cm^3;

1 slug/ft^3 = 0.5154 gm/cm^3

Force

1 newton (nt) = 10^5 dynes = 0.1020 kg = 0.2248 lb

1 pound weight (lb) = 4.448 nt = 0.4536 kg = 32.17 poundals

1 kilogram weight (kg) = 2.205 lb = 9.807 nt

1 U.S. short ton = 2000 lb : 1 long ton = 2240 lb;

1 metric ton = 2205 lb

Energy

1 joule = 1 nt m = 10^7 ergs = 0.7376 ft lb = 0.2389 cal = 9.481×10^{-4} Btu

1 ft lbwt = 1.356 joules = 0.3239 cal = 1.285×10^{3} Btu

1 calorie (cal) = 4.186 joules = 3.087 ft lb = 3.968×10^{-3} Btu

1 Btu (British thermal unit) = 778 ft lb = 1055 joules = 0.293 watt hr

1 kilowatt hour (kw hr) = 3.60×10^{6} joules = 860.0 kcal = 3413 Btu

1 electron volt (ev) = 1.602×10^{-19} joule

Power

1 watt = 1 joule/sec = 10^7 ergs/sec = 0.2389 cal/sec

1 horsepower (hp) = 550 ft lb/sec = 33,000 ft lb/min = 745.7 watts

1 kilowatt (kw) = 1.341 hp = 737.6 ft lb/sec = 0.9483 Btu/sec

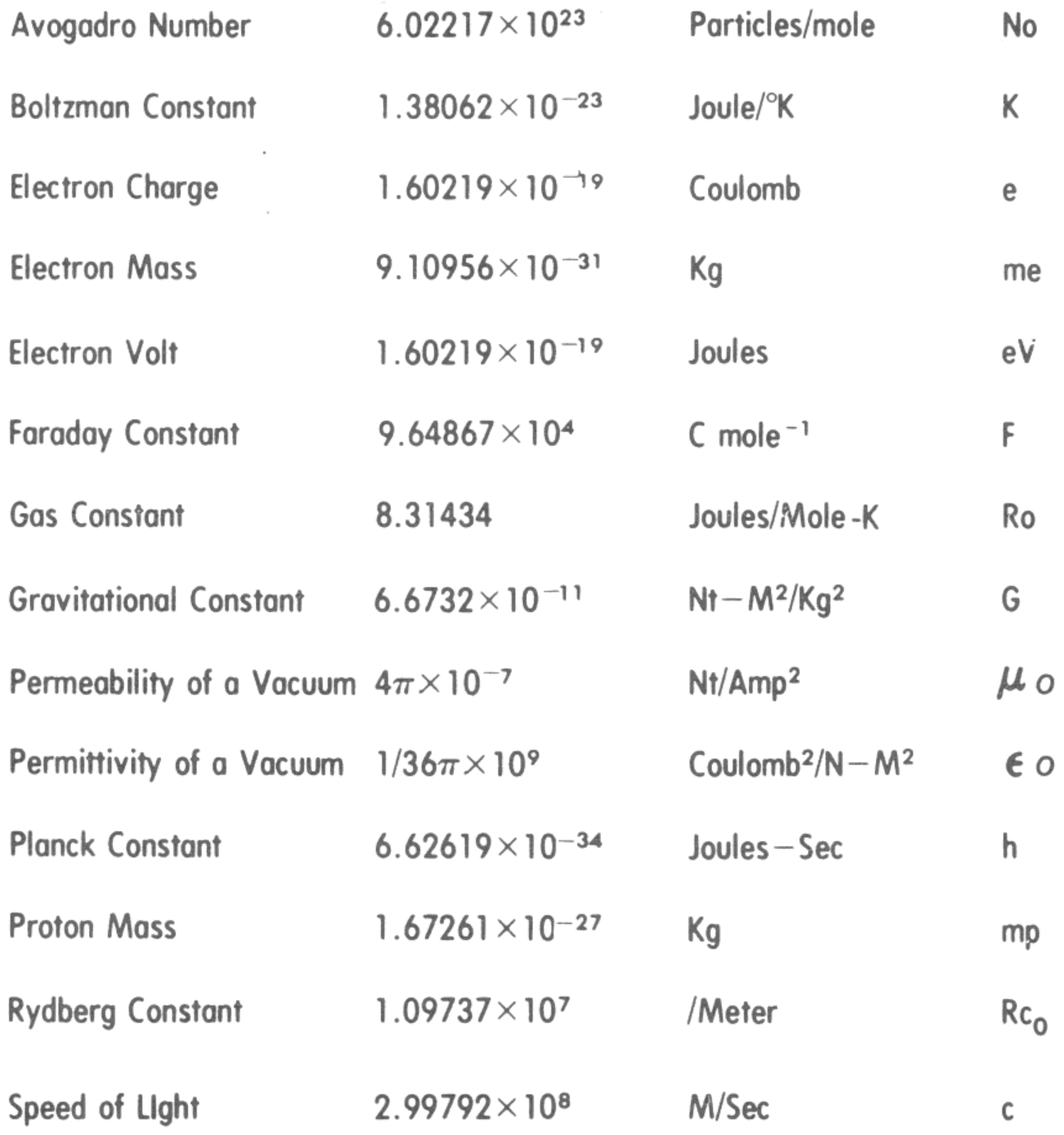
Pressure

1 nt/m^2 = 10 dynes/cm^2 = 9.869×10^{-6} atmosphere = 2.089×10^{-2} lb/ft^2

1 lbwt/in^2 = 6895 nt/m^2 = 5.171 cm mercury = 27.68 in. water

1 atmosphere (atm) = 1.013 c 10^5 nt/m^2 = 1.013×10^6 dynes/cm^2
= 14.70 lb/in^2 = 76 cm mercury = 406.8 in. water

Physical Constants



Physical Constant	Value	Units	Symbol
Avogadro Number	6.02217×10^{23}	Particles/mole	No
Boltzman Constant	1.38062×10^{-23}	Joule/°K	K
Electron Charge	1.60219×10^{-19}	Coulomb	e
Electron Mass	9.10956×10^{-31}	Kg	me
Electron Volt	1.60219×10^{-19}	Joules	eV
Faraday Constant	9.64867×10^{4}	C mole^{-1}	F
Gas Constant	8.31434	Joules/Mole-K	Ro
Gravitational Constant	6.6732×10^{-11}	$\text{Nt}-\text{M}^2/\text{Kg}^2$	G
Permeability of a Vacuum	$4\pi\times10^{-7}$	Nt/Amp^2	μo
Permittivity of a Vacuum	$1/36\pi\times10^{9}$	$\text{Coulomb}^2/\text{N}-\text{M}^2$	ϵo
Planck Constant	6.62619×10^{-34}	Joules – Sec	h
Proton Mass	1.67261×10^{-27}	Kg	mp
Rydberg Constant	1.09737×10^{7}	/Meter	Rc_0
Speed of Llght	2.99792×10^{8}	M/Sec	c

Prefixes For Power of Ten

Prefix	Symbol	Multiple
Tera	T	10^{12}
Giga	G	10^{9}
Mega	M	10^{6}
Kilo	k	10^{3}
Hecto	h	10^{2}
Deka	da	10
Deci	d	10^{-1}
Centi	c	10^{-2}
Milli	m	10-^{3}
Micro	u	10-^{6}
Nano	n	10^{-9}
Pico	p	10^{-12}
Femto	f	10-^{15}
Atto	a	10-^{18}